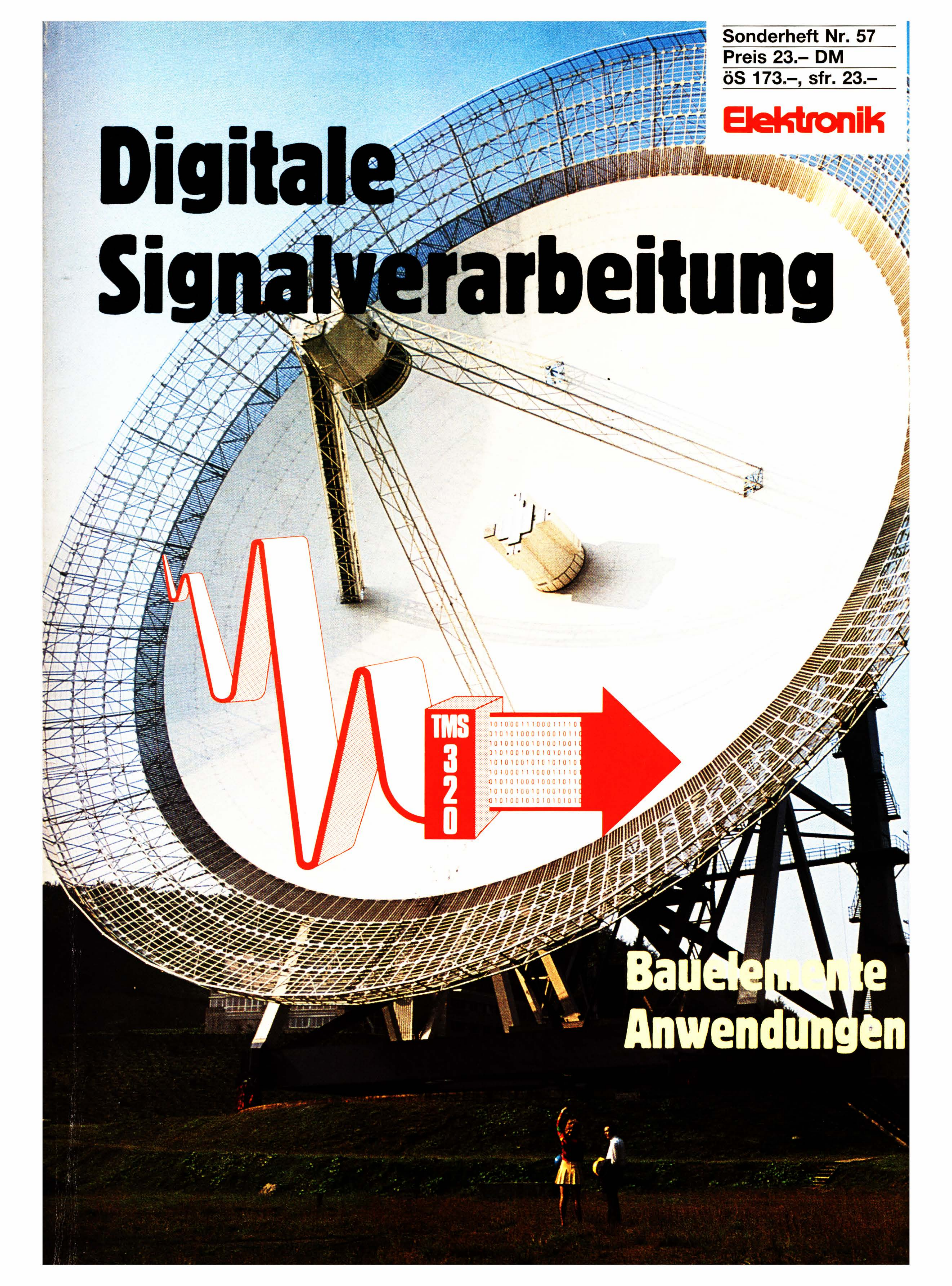


Sonderheft Nr. 57
Preis 23.– DM
öS 173.–, sfr. 23.–
Elektronik
Digitale Signalverarbeitung
TMS 320
Bauelemente
Anwendungen

Mit MAC plus (oder einfach: ADSP 1110) eröffnet Analog Devices der digitalen Signalverarbeitung völlig neue Dimensionen. Denn MAC plus – ein 16 x 16 Bit-CMOS-Multiplizierer/Akkumulator mit Single-Port-Technik – ist preiswerter als aufwendige Signalprozessoren aber entschieden komfortabler als ein herkömmlicher digitaler Multiplizierer/Akkumulator (MAC).

Obwohl MAC plus für erstaunlich wenig Geld zu haben ist, enthält und bietet er weitaus mehr, als Sie von früheren Bausteinen erwarten durften:

- 16 x 16-Bit Multiplizierer für Zweier-Komplement und Betragszahlen
- 40-Bit-Addierer mit Rundung an zwei wählbaren Stellen
- Sättigungslogik mit Überlaufanzeige
- Schieberegister
- Registeraustausch für „Double Precision"-Multiplikationen (32 x 32 Bit)
- Geringe Leistungsaufnahme durch neue „High Speed"-CMOS-Technik

Applikationen
FIR- und IIR-Filter, schnelle Fourier-Transformation FFT (1024 Punkte in 10 ms), schnelle Vektorrechnung.

Sprechen Sie mit AD –
wenn digitale Signalverarbeitung Ihr Thema ist. Wir halten Sie technologisch auf dem Laufenden.

ANALOG DEVICES



MAC plus

Die neue Formel in der digitalen Signalverarbeitung

Analog Devices GmbH, Edelsbergstraße 8, 8000 München 21, Tel. (089) 57005-0, Tlx. 523712
Technische Büros und Vertretungen: 1000 Berlin, Tel. (030) 316441, Tlx. 183326
2110 Buchholz/Hmbg., Tel. (04187) 381, Tlx. 2189375 · 5000 Köln, Tel. (0221) 686006 · 7500 Karlsruhe, Tel. (0721) 616075, Tlx. 7825871
A-1090 Wien, Tel. (0222) 235555-0, Tlx. 134759 · CH-1201 Genf, Tel. (022) 315760, Tlx. 289096

Vorwort

Während die Rechner- und Steuerungstechnik ohne digitale Verfahren undenkbar ist, schien die Verarbeitung elektrischer Signale insbesondere in der Niederfrequenz- und Nachrichtentechnik bisher eine Domäne der Analogtechnik zu sein. Daß man diese Signale auch digital verarbeiten kann, ist zwar lange bekannt und ließ sich mit Hilfe komplizierter mathematischer Methoden auch erfassen, eine breite Anwendung scheiterte jedoch bis vor einigen Jahren am unverhältnismäßig hohen technischen Aufwand.

Nachdem es mit dem inzwischen erreichten Stand der Halbleitertechnik möglich ist, außerordentlich komplexe Schaltungen zu integrieren, entfällt diese Einschränkung. Heute stehen speziell für die digitale Signalverarbeitung konzipierte Prozessoren als Halbleiterbauelemente zur Verfügung.

Mit diesen ICs lassen sich Systeme aufbauen, die analoge Signale digital verarbeiten, und zwar in vielen Fällen besser als mit den herkömmlichen Verfahren. Ein weiterer wichtiger Punkt unterscheidet die digitale Signalverarbeitung von der konventionellen Technik: weil beim digitalen Signalprozessor das Programm die Art der Verarbeitung festlegt, läßt sich ein solches System ohne Verändern der Hardware modifizieren.

Während viele technische Probleme mit der Verfügbarkeit der Bausteine gelöst sind, scheitert eine praktische Anwendung oft daran, daß viele Entwickler zu wenig Erfahrung in dieser noch jungen Technik haben. Hier soll das vorliegende Sonderheft der ELEKTRONIK eine Hilfe sein, indem es Informationen über die Signalprozessor-Bauelemente verschiedener Hersteller gibt und typischen Anwendungen zeigt.

Die Redaktion

Inhalt

Titelbild:
Radio-Teleskop bei Effelsberg/Eifel
(G. Kalt/Mauritius/Texas Instruments)

1984

Franzis-Verlag GmbH, Karlstraße 37–41, 8000 München 2. Bearbeitet von der Redaktion der Zeitschrift ELEKTRONIK. Für den Text verantwortlich: Dipl.-Ing. Peter von Bechen.

ISSN 0170-0898

Druck: Franzis-Druck GmbH, München. Printed in Germany. Imprimé en Allemagne.
ZV-Artikel-Nr. 57021 · F/ZV/1184/1050/10'

Dr.-Ing. H.-J. Kolb

Prozessorkonzepte zur digitalen Signalverarbeitung

Die digitale Signalverarbeitung stellt hohe Anforderungen an Verarbeitungsleistung und Flexibilität von Prozessoren. Entsprechend den Anforderungen an die Systeme gibt es verschiedene Formen der Realisierung, die von der Verwendung der neuen Signalprozessorchips bis hin zum Einsatz sehr universeller und aufwendiger Prozeßrechner reichen. Der folgende Beitrag zeigt in einem Überblick die möglichen Konzepte und momentan verfügbaren Systeme. In einer kurzen Einführung in typische Algorithmen der Signalverarbeitung werden die Anforderungen in anschaulicher Weise verdeutlicht.

1 Aufgaben und Methoden der digitalen Signalverarbeitung

Digitale Signalverarbeitung ist ein relativ junges eigenständiges Fachgebiet. Entwicklung und Erforschung der meisten Verfahren und Algorithmen wurde erst in den letzten Jahren einigermaßen abgeschlossen. Die Methoden und Verfahren der digitalen Signalverarbeitung setzt man im wesentlichen zur Lösung der folgenden zwei Aufgaben immer häufiger ein:

- Entwicklung von komplexen Systemen, die mit analogen Mitteln nicht realisierbar sind,
- Digitalisierung von Systemen, die bisher mit analogen Mitteln realisiert wurden. Hierdurch ergeben sich die bekannten Vorteile der Digitaltechnik, die durch die Stichworte vereinfachte Produktion und Wartung, Zuverlässigkeit, Stabilität, Programmierbarkeit usw. beschrieben werden.

Auffallend ist, daß es gerade viele nicht nachrichtentechnische Bereiche sind, in denen die Methoden erfolgreich einzusetzen sind. Beispiele finden sich hierfür in der Medizintechnik, Verfahrenstechnik, Regelungstechnik usw. Hieraus ergibt sich für viele Anwender die Notwendigkeit, sich in bisher unbekannte Methoden und Verfahren einzuarbeiten. Im folgenden werden deshalb einige typische Algorithmen vorgestellt, um die Grundlage für die nachfolgenden Leistungsvergleiche zu schaffen. Es ist nicht das Ziel dieser Arbeit, eine komplette systemtheoretische Beschreibung zu geben. Darüber existiert umfangreiche Literatur [1, 2].

Ausgangspunkt für alle digitalen Verfahren ist ein Signal, das durch eine Zahlenfolge dargestellt wird. Bei der Verarbeitung analoger Größen gewinnt man diese Zahlenfolge durch die Abtastung eines analogen Signales mit anschließender Analog-Digital-Umsetzung. Diese Abtastung erfolgt in der Regel fortlaufend, ohne Unterbrechung, zu äquidistanten Zeitpunkten. Fehler, die bereits hier, vor der eigentlichen Verarbeitung, entstehen (z. B. Aliasing, Quantisierungsfehler), werden hier nicht weiter diskutiert. Sie waren und sind Gegenstand verschiedener Untersuchungen, Ergebnisse können der Literatur entnommen werden [1]. *Bild 1* zeigt schematisch die Aufgaben und Funktionsblöcke eines digitalen Systems zur Verarbeitung analoger, eindimensionaler Signale. Das bandbegrenzte Signal v(t) wird abgetastet und anschließend von analoger in digitale Form umgesetzt. Der A/D-Umsetzer liefert an seinem Ausgang die Folge v(k), die im Signalprozessor der eigentlichen Verarbeitung unterworfen wird. Das Ergebnis dieser Verarbeitung ist in der Regel eine Zahlenfolge y(k). In vielen Fällen werden die Ausgangswerte y(k) im gleichen Zeitraster wie die der Eingangsfolge v(k) errechnet. Benötigt man die Ergebnisse der Verarbeitung wiederum als analoges Signal, so ist eine Digital-Analog-

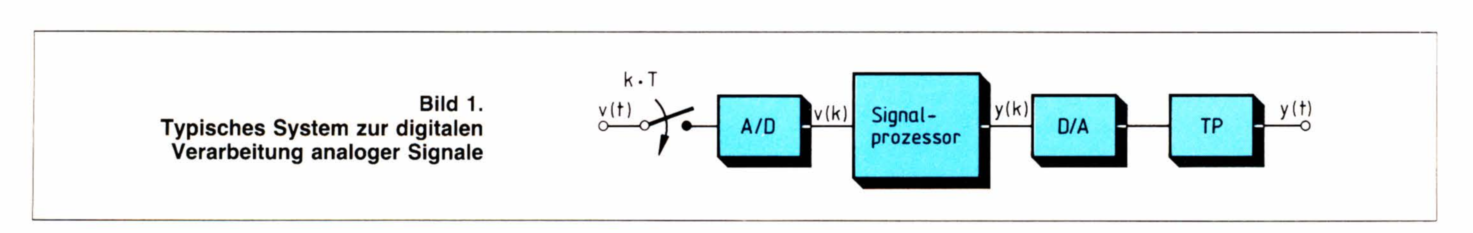


Bild 1.
Typisches System zur digitalen Verarbeitung analoger Signale

Umsetzung mit anschließender Glättung des Zeitsignales in einem geeigneten Tiefpaß erforderlich.

Innerhalb der Signalverarbeitung bilden die Methoden zur digitalen Filterung eine wesentliche Gruppe. Man unterscheidet rekursive und nichtrekursive Verfahren. *Bild 2* zeigt das Blockschaltbild eines nichtrekursiven Filters in der direkten Struktur. Bei den rekursiven Verfahren werden vergangene Filterausgangswerte y(k-1), y(k-2), ... zur Berechnung des aktuellen Ausgangswertes y(k) herangezogen. Im Gegensatz zur nichtrekursiven Filterung gibt es hier eine Vielzahl verschiedener, zweckmäßiger Strukturen und Realisierungsmöglichkeiten. Diese Vielfalt resultiert aus unterschiedlichem Verhalten bezüglich der Fehlerfortpflanzung, der Stabilität sowie des numerischen Aufwandes bei der Berechnung der Filterausgangswerte y(k). Eine sehr häufig verwendete Methode zur Realisierung rekursiver Filter ist die Kaskaden- oder Parallelanordnung von Blökken 2. Grades. *Bild 3* zeigt ein Beispiel für die Realisierung eines Blockes 2. Grades in der 1. kanonischen Form. Filter höheren Grades werden durch Kaskadierung oder Parallelanordnung dieser Basisbausteine realisiert.

An den gegebenen Beispielen aus dem Bereich der digitalen Filterung ist ersichtlich, daß die zentrale Rechenoperation hier die Berechnung des Skalarproduktes zweier Vektoren ist. Auch andere Algorithmen, wie z. B. die Berechnung von Korrelationsfunktionen, enthalten als wesentlichen Teil diese Skalarproduktbildung. Auffällig ist dabei, daß die Algorithmen im Gegensatz zu anderen Datenverarbeitungsaufgaben verhältnismäßig viele Multiplikationen enthalten. Wesentlich ist darüber hinaus die Verwaltung der Zustandsgrößen im Datenspeicher. Hieraus ergibt sich die Forderung nach bestimmten, bei Universalrechnern unüblichen Adressierungsarten (z. B. Modulo-Adressierung). Diese Ergebnisse finden sich auch bei praktisch allen anderen Algorithmen innerhalb der Signalverarbeitung.

Eine weitere sehr wichtige Operation in der digitalen Signalverarbeitung ist die diskrete Fouriertransformation. *Bild 4* zeigt den Signalflußgraph zur Berechnung der DFT für eine Eingangsfolge mit komplexen Elementen der Länge 8. Ähnlich wie bei der rekursiven Filterung gibt es auch hier eine Vielzahl verschiedener Algorithmen zur möglichst schnellen Berechnung dieser Transformation. Bild 4 zeigt als Beispiel „decimation in frequency – radix 2". Wie im Falle der digitalen Filterung, wird auch hier die große Anzahl von Multiplikationen deutlich.

Eine wesentliche Frage bei der Realisierung der unterschiedlichen Algorithmen ist die Spezifikation der Arithmetik. An sich wäre es aus der Forderung nach einer hohen Dynamik oder einer hohen Genauigkeit und damit einem großen Signal-Störabstand wünschenswert, Gleitkommaarithmetik mit möglichst großer Mantissen- und Exponentenwortlänge zu verwenden. Jedoch scheitert die Erfüllung dieses Wunsches in den meisten Fällen an der Forderung nach möglichst geringem Aufwand und/oder nach einer hohen Rechenleistung. Häufig ist aber mit 16-Bit-Festkommaarithmetik ein guter Kompromiß zwischen den Wünschen nach hoher Dynamik und günstigem Signal-Störabstand einerseits und minimalem Aufwand andererseits zu erreichen.

Bereits an diesen einfachen ausgewählten Beispielen lassen sich die aus den Algorithmen resultierenden Forderungen erkennen, die an Signalprozessoren zu stellen sind. Eine zusammenfassende Darstellung ist in *Tabelle 1* angegeben.

Tabelle 1. Forderungen an Signalprozessoren

Daten:	ausreichende Wortlänge (16 Bit)
Koeffizienten:	ausreichende Wortlänge
Arithmetik:	schnelle Multiplikation, schnelle Akkumulation
Architektur:	schneller Datentransfer vom Speicher in die Arithmetikeinheiten möglichst große Speicher für Programme, Daten und Koeffizienten

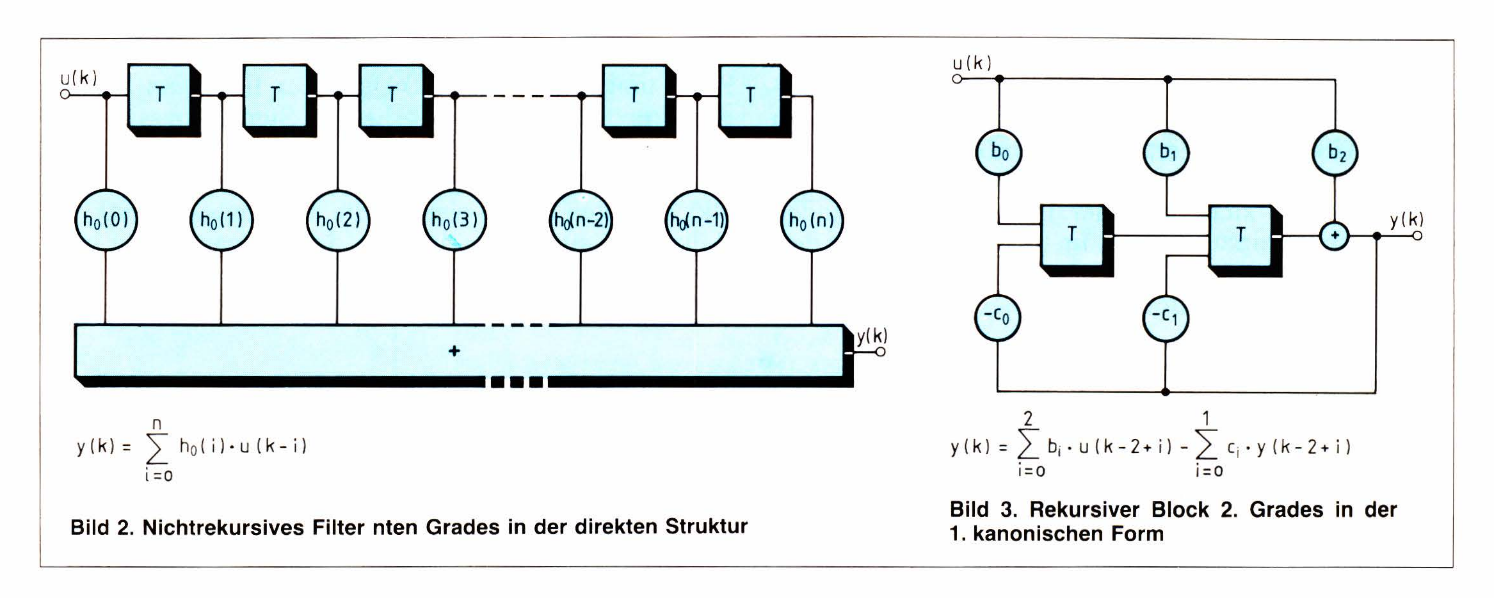


Bild 2. Nichtrekursives Filter nten Grades in der direkten Struktur

Bild 3. Rekursiver Block 2. Grades in der 1. kanonischen Form

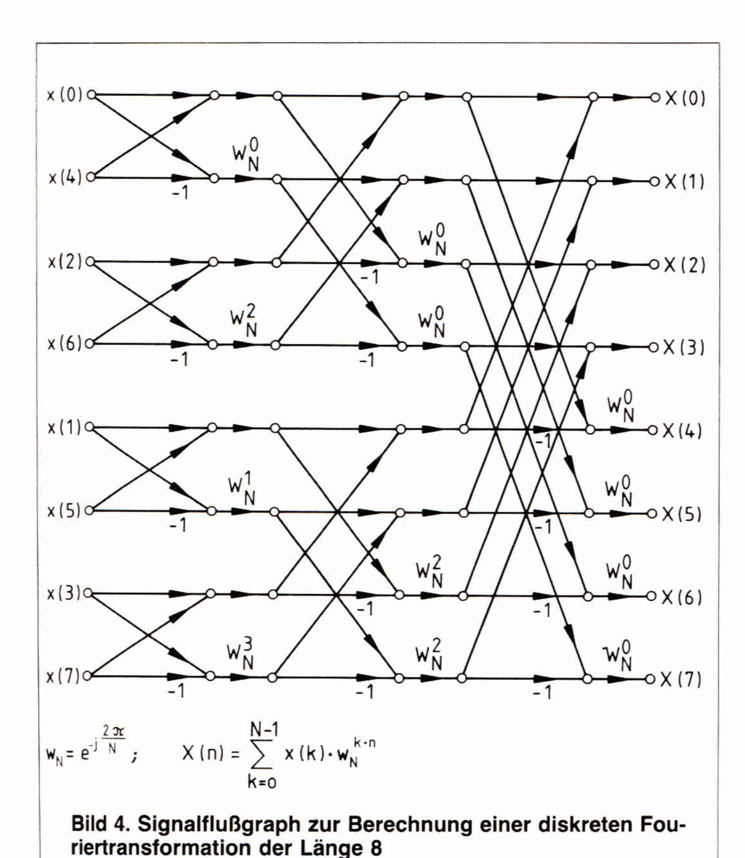


Bild 4. Signalflußgraph zur Berechnung einer diskreten Fouriertransformation der Länge 8

2 Verschiedene Konzepte integrierter Signalprozessoren

Die wachsende Anzahl von Aufgaben und Problemen, die mit den Methoden der Signalverarbeitung bearbeitet und gelöst werden, führte zur Entwicklung verschiedener Signalprozessoren oder Bausteinfamilien zur möglichst guten Unterstützung der Systementwickler. Man kann hierbei verschiedene Konzepte und Strategien bei den Herstellern erkennen, die in *Tabelle 2* neben den einzelnen aufgeführten Bausteinen zusammenfassend dargestellt sind.

AMD verfolgt das offensichtliche Ziel, die Bausteinfamilien zur Vereinfachung der Realisierung leistungsfähiger flexibler mikroprogrammierter Systeme weiter zu ergänzen und auszubauen. Hervorzuheben ist die sehr angenehme und häufig entscheidende Eigenschaft der Kaskadierbarkeit der einzelnen Slice-Elemente zu beliebigen Wortlängen, die sich wie bei den Bit-Slice-Elementen der Bausteinfamilie Am2900 auch bei der 8-Bit-Register/ALU Am29501 findet. Dieses Konzept kann nicht mehr ohne weiteres auf die Realisierung von Parallelmultiplizierer-Bausteinen übertragen werden. AMD hat sich damit auch zur Entwicklung eines 16-Bit-Parallelmultiplizierer-Bausteines entschlossen, der dem vergleichbaren entsprechenden Multiplizierer von TRW sehr ähnlich ist.

Auch die Bausteine der Firma TRW sind für Systeme höchster Leistungsfähigkeit konzipiert. Man findet Multiplizierer sowie Multiplizierer mit integrierten Akkumulatoren für unterschiedliche Wortlängen. Sind Wortlängen von weniger als 16 Bit für die Verarbeitung von z. B. Audio-Signalen häufig unzureichend, so gilt dies nicht für die Verarbeitung von Videosignalen. Mit den verfügbaren, wie auch den angekündigten Bausteinen dieser Familien ist eine Video-Echtzeitverarbeitung bei Multiplikationszeiten von 50...100 ns bereits möglich. Auch bei TRW ist die Tendenz festzustellen, das TTL-Programm durch schnelle, weniger Leistung verbrauchende MOS-Bausteine zu ergänzen. Angekündigt ist ein 16-Bit-Multiplizierer/Akkumulator in CMOS-Technik mit 100 ns Multiplikationszeit und 300 mW Verlustleistung. Eine weitere sehr interessante Ergänzung ist

Tabelle 2. Zusammenstellung derzeit bekannter Signalprozessorbausteine (Stand Mitte 1982)

Firma	Chip	Beschreibung	Bemerkungen
AMD	Am29501 Am29516/17 Am29520/21 Am29540	Register/ALU 8-Bit-Slice Multiplizierer 16 Bit Multilevel-Pipeline-Register FFT-Adreß-Sequencer	Unterstützung mikroprogrammierter Systeme, die vorwiegend aus MSI-Bausteinen aufgebaut sind Am29501 und Am29540 lieferbar 4.Qu.82
AMI	S2811 S2814 S2815 S2816	universeller Signalprozessor FFT programmierter S2811 Filter programmierter S2811 Echoentzerrer progr. S2811	Basischip ist der S2811 Hierauf aufbauend werden verschiedene Chips mit festen Programmen angeboten. Verfügbar S2811, S2814 Einzelstückpreis ca. 750.– DM
Bell	DSP	universeller Signalprozessor	DSP ist nicht auf dem freien Markt erhältlich
Intel	2920	universeller Signalprozessor	Baustein mit Analogein- und -ausgabemöglichkeit; ohne Multiplizierer Einzelstückpreis ca. 400.– DM
ITT	MA1000	universeller Signalprozessor	in der Entwicklung
NEC	7720	universeller Signalprozessor	EPROM-Version erhältlich Einzelstückpreis ca. 1000.– DM
TI	TMS320	universeller Signalprozessor	
TRW	MPY-8/12/16HJ TDC1008/9/10J TDC1022	schnelle Festkommamultiplizierer Multiplizierer-Akkumulatoren Gleitkomma-Addierer 16+6 Bit	Die Bausteine sind konzipiert, um in speziellen Prozessoren höchster Leistung eingesetzt zu werden (Videosignalv.)

mit Bausteinen zur schnellen Gleitkommaverarbeitung gegeben. Das erste Produkt in diesem Bereich ist ein 22-Bit-Gleitkomma-Addierer mit 16-Bit-Mantisse und 6-Bit-Exponent. Auch hier sind Verarbeitungszeiten von weniger als 100 ns angegeben. Mit der Verwendung der Gleitkomma-Arithmetik ergibt sich eine wesentliche Erhöhung der Dynamik in der digitalen Verarbeitung. Weiterhin entfällt die insbesondere bei rekursiven Algorithmen sehr aufwendige Skalierung. Dieses Problem stellt sich ähnlich wie bei der Programmierung des Analogrechners. Auch in der digitalen Verarbeitung ist es bei Verwendung der Festkomma-Arithmetik anzustreben, die verschiedenen Arithmetikeinheiten möglichst voll auszusteuern, um trotz des unvermeidlichen Rundungsrauschens einen möglichst großen Signal-Störabstand zu erhalten.

Die verbleibende Gruppe von Einchip-Signalprozessoren verschiedener Hersteller ist konzipiert, um in sehr viel kompakteren Systemen eingesetzt zu werden. Hier war das Ziel nicht nur eine hohe Leistungsfähigkeit, sondern auch, möglichst alle benötigten Funktionen eines kompletten Prozessors auf einem Chip zu integrieren. Die erreichten Ergebnisse zeigt der folgende Abschnitt.

3 Einchip-Signalprozessoren

Grundsätzlich ist es interessant und aufschlußreich, die Leistungsmerkmale der verschiedenen Signalprozessoren und Prozessorkonzepte zu vergleichen. Hierbei muß darauf hingewiesen werden, daß ein derartiger Vergleich nur eine grobe Übersicht bieten kann. Im Einzelfall können zusätzliche Argumente entscheidende Bedeutung erlangen. Insbesondere kann der Vergleich in *Tabelle 3* den Einfluß der verschiedenen Arithmetik-Realisierungen auf das Fehlerverhalten im Ausgangssignal nur sehr unzureichend erläutern, obwohl gerade dieser Fehlereinfluß häufig eine wichtige Rolle spielt. Hierbei sind Eigenschaften wie z. B. Akkumulatorwortlänge, Wortlängenverkürzung im Anschluß an die Multiplikation oder Akkumulation insbesondere bei rekursiven Algorithmen von entscheidender Bedeutung. Weiterhin ist eine Einbeziehung der Bausteinfamilien von AMD und TRW in diese Übersicht nicht möglich, da man für diese wegen der vielen Freiheitsgrade bei der Wahl der Prozessorarchitektur keine typischen Signalprozessorstrukturen angeben kann, die in diesem Vergleich angeführt werden könnten.

Tabelle 3 zeigt, daß mit den vorhandenen Signalprozessoren die Möglichkeit gegeben ist, typische Algorithmen der Signalverarbeitung für Signale im Audio-Bereich zu realisieren. Weiterhin zeigen sich deutliche Unterschiede in der Leistungsfähigkeit der Prozessorchips. Der am längsten verfügbare Prozessor 2920 (Intel) ist der einzige ohne einen integrierten Parallelmultiplizierer. Multiplikationen werden hier mit hohem Zeitaufwand mittels Software realisiert. Dafür enthält dieser Baustein einen 9-Bit-A/D- und D/A-Umsetzer, was das Interface-Problem in einer analogen Umgebung ent-

Tabelle 3. Leistungsmerkmale der Einchip-Signalprozessoren (Stand: Juni 1982)

* = entweder keine Angaben verfügbar oder Vergleich auf Grund einer starken Abhängigkeit der Rechenzeit von den Koeffizienten nicht sinnvoll

Merkmal	AMI S2811	Bell DSP	Intel 2920	ITT MA1000	NEC 7720	Texas TMS320
Technologie	V-MOS	NMOS	NMOS	NMOS	NMOS	NMOS
Anzahl der Anschlüsse	28	40	28	*	28	40
Programm-speicher	250x17 Bit ROM	1024x16 Bit ROM	192x24 Bit EPROM	512x26 Bit ROM	512x23 Bit ROM/EPROM	1,5k-4kx16 Bit ROM/Off chip
RAM	128x16 Bit	128x20 Bit	40x25 Bit	32x8-Bit-Koeff. 128x24 Bit-Dat.	128x16 Bit	144x16 Bit
ROM (Daten)	128x16 Bit	—	—	128x8 Bit	512x13 Bit	—
Akkumulator Wortlänge (in Bit)	16	40	25	24	16	32
Hardware-multiplizierer	12x12-16 Bit	20x16-35 Bit	Software	16x8 Bit	16x16-31 Bit	16x16-3 Bit
EPROM-Version verfügbar?	nein	nein	ja	nein	ja	nein
Befehlszyklus-zeit (ns)	300	800	400	250	250	200
Operation pro Zyklus	multipliz. addieren speichern	multipliz. addieren	shift addieren speichern	multipliz. addieren	multipliz. addieren speichern	multipliz. addieren
rekursiver Block 2. Grades	2,1 µs	4,0 µs	*	*	2,5 µs	*
FFT 32 Punkte komplex	1500 µs	*	*	*	700 µs	*
Versorgung	+5 V/1 W	+5 V/1,5 W	+5 V, −5 V/0,8 W	+5 V	+5 V/0,9 W	+5 V/0,95 W

scheidend vereinfachen kann. Es ist jedoch zu beachten, daß diese 9 Bit für viele Anwendungen keinen ausreichenden Signal-Störabstand gewährleisten.

Der Signalprozessor S2811 der Firma AMI wurde primär zur Kopplung an den 8-Bit-Mikroprozessor 6800 konzipiert. Wie bei allen anderen Signalprozessoren ist jedoch auch hier der autonome Betrieb möglich. Der S2811 enthält einen 12x12-Bit-Multiplizierer neben einer arithmetischen Einheit mit 16 Bit. Für viele Algorithmen stellt die hieraus resultierende maximale Wortlänge von 12 Bit für Daten und Koeffizienten bei der Multiplikation eine starke Beschränkung dar. Weiterhin ist der Prozessor S2811 nicht in EPROM-Version verfügbar, was die Entwicklung von Software erschwert und die Anwendungen auf große Stückzahlen beschränkt. AMI begegnet diesem Problem durch die Entwicklung vorprogrammierter Prozessoren für bestimmte häufig auftretende Problemstellungen. Ein Beispiel hierfür ist der bereits verfügbare Baustein S2814 zur Berechnung der FFT; weitere Bausteine sind angekündigt.

Der in der Entwicklung befindliche Prozessor MA 1000 von ITT weist im Vergleich zu den leistungsfähigsten Bausteinen der Firmen Bell, NEC und Texas Instruments starke Beschränkungen auf. Sein Multiplizierer ist im Vergleich deutlich leistungsschwächer. Es muß bezweifelt werden, ob sich dieser Baustein nach der Fertigentwicklung auf dem allgemeinen Markt durchsetzen kann.

Sehr leistungsfähig sind die gleichzeitig auch jüngsten Bausteine der Firmen Bell, NEC und Texas Instruments. Diese erlauben die Realisierung umfangreicher und auch aufwendiger Verfahren der Audiosignalverarbeitung. Insbesondere für die Prozessoren 7720 (NEC) und DSP (Bell) gibt es eine große Zahl von Veröffentlichungen zu typischen Anwendungsbeispielen aus den Bereichen der Datenübertragung und der Sprachverarbeitung. Der angekündigte Baustein TMS320 (ausführliche Beschreibung in diesem Heft) weist als Besonderheit eine 32-Bit-ALU auf. Obwohl der Baustein 7720 von NEC eindeutig die momentan größte Leistungsfähigkeit besitzt, weist der DSP von Bell Vorteile auf, wenn man den Fehlereinfluß durch die begrenzte Wortlänge berücksichtigt. Er besitzt als einziger Prozessor eine Wortlänge von 20 Bit für die Daten, was ihn zum Beispiel für die Verarbeitung sehr hochwertiger Signale im Audio-Bereich geeignet macht. Auch der Akkumulator ist im Gegensatz zum NEC-Chip in der Lage, die volle Wortlänge des Multiplikationsergebnisses in einem Zyklus zu verarbeiten. Damit werden Fehler, die durch laufende Wortlängenverkürzung entstehen, minimiert. Weiterhin besitzt der DSP als einziger Chip besondere Vorkehrungen zur Ausführung einer Rundung bei der Wortlängenverkürzung, was die Fehlerwirkung weiter reduziert. Der DSP wird von Bell nicht auf dem freien Markt verkauft. Eine ähnliche Politik ist auch bei anderen Herstellern festzustellen; z. B. verfügt NEC über noch leistungsfähigere Prozessoren, die nur in eigenen Produkten verwendet werden.

Tabelle 4. Rechenzeiten von 16-Bit-Mikroprozessoren bei typischen Signalverarbeitungsalgorithmen

(* = Bezieht sich auf ein System aus 8085, NEC 7720 und DMA-Controller)

Prozessor	rekursives Filter 10. Grades	nichtrekursives Filter 64. Grades	FFT 512 reelle Punkte
Texas 9900 3 MHz	1730 µs	4651 µs	—
Intel 8086 10 MHz	527 µs	1475 µs	340 ms
Zilog Z8000 8 MHz	365 µs	800 µs	174 ms
NEC 7720 8 MHz	12,5 µs	66 µs	etwa 20 ms*

4 Realisierung der Signalverarbeitungs-Algorithmen auf universellen Rechnern und Prozessoren

Bei der Betrachtung von Systemen zur Signalverarbeitung sollte darauf hingewiesen werden, daß der weitaus größte Teil der Nicht-Echtzeit-Signalverarbeitung, wie auch der Verarbeitung von Signalen mit geringer Bandbreite von einigen Hundert Hertz, seit langem auf Universalrechnern abgewickelt werden. Zur Verringerung der Verarbeitungszeiten auf kleineren Anlagen werden hierbei häufig Array-Prozessoren eingesetzt. Vorteil dieser Vorgehensweise ist eine sehr viel einfachere und komfortablere Programmerstellung, die gerade im Bereich der Forschung und Algorithmenentwicklung wichtiger als schnelle Echtzeitverarbeitung ist. Es kommt hinzu, daß die Effekte der begrenzten Wortlänge, bedingt durch die hier üblicherweise angewandte Gleitkommaarithmetik meist unberücksichtigt bleiben können. Nachteilig ist bei vielen Aufgaben der bei einer relativ geringen Verarbeitungsleistung hohe Aufwand dieser Systeme.

Andererseits darf nicht übersehen werden, daß selbstverständlich die Möglichkeit besteht, die Algorithmen auf jedem universellen Mikroprozessor zu implementieren. *Tabelle 4* zeigt Ergebnisse, die am Lehrstuhl für Nachrichtentechnik der Universität Erlangen in insgesamt drei Diplomarbeiten an verschiedenen Mikroprozessorsystemen ermittel wurden. Es ist darauf hinzuweisen, daß es sich hierbei um Messungen an Programmen handelt, deren Rechenzeitbedarf minimiert wurde. Die einzelnen Mikroprozessorsysteme werden nicht durch zusätzliche Hardware, wie z. B. Multiplizierer unterstützt. Die gesamten Programme sind für 16-Bit-Festkommaarithmetik realisiert. Damit ist ein unmittelbarer Vergleich mit den Signalprozessoren NEC 7720, Bell DSP und dem Texas TMS320 möglich. Zum besseren Vergleich der Leistungsfähigkeit sind in der Tabelle 4 zusätzlich Rechenzeiten für den Signalprozessor

NEC 7720 angegeben. Die Ergebnisse zeigen eine deutliche Überlegenheit der speziellen Signalprozessoren gegenüber den universellen Mikroprozessoren. Wichtigster Grund hierfür ist die spezialisierte Architektur der Signalprozessoren. Man hatte hier die Freiheit, mehr Wege auf dem Chip zu realisieren, um die Arithmetikeinheiten schneller mit Daten versorgen zu können. Üblich sind hier mindestens zwei Datenbusse. Bei den universellen Mikroprozessoren steht dagegen die Vereinheitlichung des Bus-Systems im Vordergrund, was dann in der Regel auf Ein-Bus-Systeme führt. Ein weiterer wesentlicher Vorteil der Signalprozessoren ist der integrierte Parallelmultiplizierer, der zu einer wesentlichen Verkürzung der Ausführungszeiten gegenüber der in Firmware realisierten Multiplikation bei modernen Mikroprozessoren beiträgt. Trotz dieses großen Leistungsunterschiedes zwischen Signalprozessor und Mikroprozessor kann der Einsatz von Mikroprozessoren bei der Verarbeitung von Signalen mit geringer Bandbreite, wie sie z. B. in der EEG- oder EKG-Verarbeitung vorliegen, durchaus sinnvoll sein. Die große Flexibilität dieser Systeme, ist in der Regel vorteilhaft. Weiterhin besteht wesentlich mehr Unterstützung bei der Systemrealisierung durch fertige Hardwarekomponenten, komplett erhältliche Systeme sowie durch den Einsatz höherer Programmiersprachen und der momentan besseren Entwicklungshilfen.

5 Einsatz spezialisierter Universal-Rechner für die Signalverarbeitung

Innerhalb der Signalverarbeitung bestehen Aufgaben, die schnell abgewickelt werden müssen, bei denen aber eine Realisierung auf Signalprozessoren wegen der dort auftretenden Einschränkungen bei Daten- und Programmspeicher nicht in Frage kommt. Es besteht somit das Problem, möglichst universelle, einfach programmierbare Rechner großer Leistungsfähigkeit zu entwikkeln. Am Lehrstuhl für Nachrichtentechnik der Universität Erlangen–Nürnberg wurde in den letzten Jahren intensiv an diesem Problem gearbeitet. Im Rahmen dieser Arbeiten wurde unter anderem ein Prozessor auf der Basis der Bit-Slice-Elemente Am2900 von AMD entwikkelt, der nun von der Firma MEDAV in Lizenz gefertigt wird. *Tabelle 5* zeigt die wesentlichen Ziele und Eigenschaften des Signalprozessors MSP82.

Bei der Realisierung des Signalprozessors MSP82 wurden die folgenden Ziele angestrebt:

- einfache Prozessorarchitektur:
  erfahrungsgemäß ist es bei einer Echtzeitanwendung von Prozessoren in der Regel nicht zu vermeiden, sehr hardwarenah zu programmieren. Um hierbei die Übersicht zu behalten, ist es zweckmäßig, eine möglichst einfache Prozessorarchitektur zu wählen.
- einfache Erweiterbarkeit der CPU zur Erhöhung der Rechenleistung:
  im Normalfall besteht die CPU eines Bit-Slice Prozessors aus der Mikroprogrammsteuerung sowie der ALU. Hiermit sind derzeit Rechenleistungen möglich, die um den Faktor 3...10 oberhalb der Rechenleistung integrierter 16-Bit-MOS-Mikroprozessoren liegen (z. B. Intel 8086, Texas 9900, Zilog Z8000, Motorola 68000). Für eine Erweiterung der arithmetischen Fähigkeiten der CPU bestehen drei Möglichkeiten:
  - Verwendung der ALU als Adreßrechner und Verlagerung der arithmetischen Operationen in ein getrenntes Rechenwerk. ALU und Zusatzprozessor werden vom zentralen Mikroprogramm gesteuert. Hiermit wird die resultierende Rechenleistung ausschließlich von der Zeit für die notwendigen Speicherzugriffe begrenzt. Man erhält eine Verarbeitungszeit, die sich ergibt als:
    $T = N \cdot TZ$,
    wobei N die Anzahl der benötigten Speicherzugriffe und TZ die Systemzykluszeit bezeichnet. Hiermit erhält man eine Erhöhung der Rechenleistung, um den Faktor 50...100 verglichen mit integrierten 16-Bit-MOS-Mikroprozessoren. Beim Vergleich mit der Rechenleistung des Signalprozessors NEC 7720 bedeutet dies noch einen Faktor von ca. zwei.
  - Kopplung von speziellen Rechenwerken wie z. B. Gleitkommavektorprozessoren über ein Dual-Port-RAM an den Zentralprozessor. Hiermit sind weitere Erhöhungen der Rechenleistung möglich, die im wesentlichen nur vom investierten Hardwareaufwand bestimmt werden.
  - Kopplung mehrerer Prozessoren, die jeder für sich Spezialhardware enthalten können. Je nach Aufgabenstellung und Problemstruktur sind verschiedene Verbindungskonzepte möglich.

Bild 5 zeigt eine Blockschaltung des Gesamtsystems mit den Erweiterungsmöglichkeiten.

Wie in der gesamten Datenverarbeitung, tritt auch in der Signalverarbeitung das Softwareproblem immer mehr in den Vordergrund. Generell fordert man in der Signalverarbeitung von der Software eine geringe Ausführungszeit sowie maximale Genauigkeit (speziell bei

Tabelle 5. Kurzbeschreibung des Signalprozessors MSP82

Architektur:	1 Datenbus 1 Adreßbus Vergrößerung der Rechenleistung durch optionelle Zusatzprozessoren
Hardware:	modularer Aufbau in Europa- und Doppeleuropakartentechnik, standardisierter Systembus, geringer Hardwareaufwand
Firmware:	einfacher Basisbefehlssatz in Anlehnung an übliche Prozeßrechner, vektororientierter Spezialbefehlssatz zur Behandlung der Signalverarbeitungsaufgaben
Software:	Betriebssystem für standalone Verwendung und für die Realisierung einer Rechnerkopplung zu einem Hostrechner Programmbibliotheken zu Signalverarbeitungsaufgaben, Makrobibliotheken zur Realisierung einer einfachen Programmierung

Festkommaarithmetik) bei möglichst geringem Hardwareaufwand. Dies führt in der Regel zu einem extrem hohen Aufwand bei der Softwareerstellung. Darüber hinaus wird die Erfüllung der genannten Zielvorstellungen häufig wesentlich vom gewählten Algorithmus beeinflußt, so daß hierdurch eine weitere Erhöhung des Aufwandes eintritt. Wesentliches Ziel bei der Realisierung des Signalprozessors MSP82 war die Reduzierung des Softwareaufwandes durch die Auswahl eines geeigneten Maschinenbefehlvorrates (Vertikalmigration) sowie die Anwendung eines geeigneten Konzeptes bei der Erstellung der Anwendersoftware auf der Maschinenbefehlsebene. Es wird ausdrücklich darauf hingewiesen, daß sich der Einsatz von höheren Programmiersprachen im vorliegenden Fall wegen des hohen Aufwandes bei der Realisierung von Übersetzerprogrammen verbietet. Es zeigte sich, daß die folgende Vorgehensweise zweckmäßig ist:

- Grundbestandteil des Prozessorbefehlssatzes ist der Basisbefehlssatz, der in Anlehnung an Befehlssätze üblicher 16-Bit-Prozeßrechner definiert wurde. Die gesamte Betriebssoftware wird aufbauend auf diesem Befehlssatz realisiert.
- Der Anwender sollte entsprechend seinen Anforderungen einen Spezialbefehlssatz definieren, der bei möglichst universellem Charakter die optimale Behandlung seiner Probleme ermöglicht. Für Anwendungen innerhalb der digitalen Signalverarbeitung konnte die hohe Effizienz eines Vektorbefehlssatzes nachgewiesen werden [11]. Dieser Spezialbefehlssatz kann je nach Rechenzeitanforderungen in unterschiedlicher Weise realisiert werden:
  - Makros auf Maschinenbefehlsebene
  - Mikroprogrammierung
  - Mikroprogrammierung mit zusätzlicher Spezialhardware (z. B. Vektorprozessor)
  - Einsatz von Spezialprozessoren, die über ein Dual-Port-RAM gekoppelt sind.
- Die Programmierung erfolgt auf der Maschinenbefehlsebene, wobei alle E/A-Operationen, Vektoroperationen usw. durch die Verwendung einer umfangreichen Makrobibliothek stark vereinfacht werden.
- Die Programmerstellung erfolgt mittels Cross-Software auf einem Hostrechner. Hierdurch besteht keine unmittelbare Notwendigkeit, umfangreiche Betriebssoftware auf dem Signalprozessor zu implementieren.
- Erprobte Software wird in Form von möglichst universellen und komfortablen Unterprogrammbibliotheken allen Anwendern zur Verfügung gestellt. Bedingt durch die Paging-Technik beim Programm- und Datenspeicher können diese Unterprogrammbibliotheken in Form von ROM-Speicherkarten zur Verfügung gestellt werden, ohne wesentliche Bereiche des Arbeitsspeichers zu belegen. Damit stehen diese Programme resident zur Verfügung, was maximalen Komfort bei maximaler Betriebssicherheit bedeutet. Bei konsequenter Anwendung der Assemblerprogrammierung mittels anwendungsspezifischer Makros entsteht eine „Quasi-HHL" (High Level Language), die dem Problemkreis stark angepaßt ist und trotz Assembler-Niveau eine

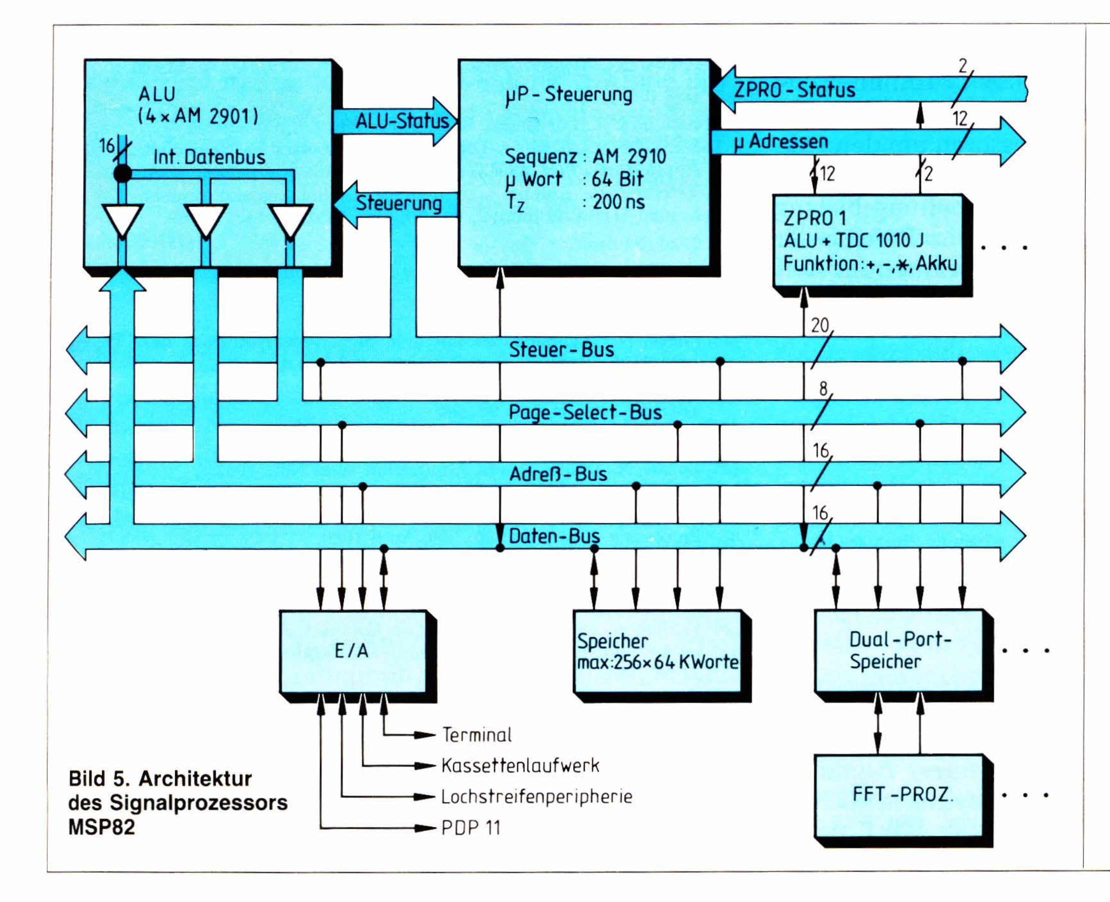
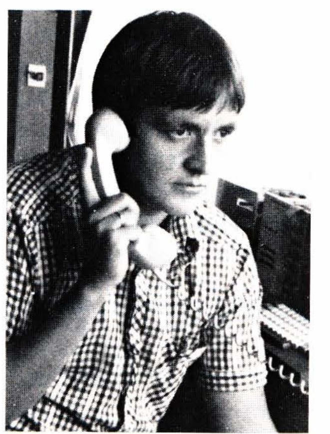


Bild 5. Architektur des Signalprozessors MSP82



Von 1972 bis 1977 studierte *Dr.-Ing. Hans-Joachim Kolb* Elektrotechnik an der Universität Erlangen-Nürnberg. Während der 5jährigen Tätigkeit am Lehrstuhl für Nachrichtentechnik war es dank der bereitwilligen Unterstützung und der stetigen Gesprächsbereitschaft seines Chefs und Doktorvaters *Prof. H. W. Schüßler* möglich, vieles über Signalverarbeitungsverfahren und die Realisierung von Systemen zur Signalverarbeitung zu lernen. Seit Juni 1982 setzt dieses Wissen in der eigenen Firma MEDAV-Meßdatenverarbeitung GmbH projektbezogen und praxisnah ein.

gute Strukturierung und Wartung der gesamten Software ermöglicht.

Mit einer CPU, bestehend aus Mikroprogrammsteuerung, ALU und Vektorprozessor (je eine Doppeleuropakarte) ergeben sich die in *Tabelle 6* angegebenen Leistungsmerkmale. Alle Rechenzeiten wurden bei einer Programmierung des MSP82 auf Maschinenbefehlsebene in 16-Bit-Festkommaarithmetik ermittelt. Die Rechenleistungen werden hier nicht durch die Verbindung eines langsamen Zentralprozessors mit sehr leistungsfähiger Spezialhardware erzielt. Wesentlich ist vielmehr, daß die gesamten Operationen mit einer schnellen, universellen CPU realisiert werden. Diese Vorgehensweise ergibt für den Benutzer den Vorteil einer einfacheren, übersichtlichen Softwareerstellung. Eine weitere Erhöhung der Rechenleistung mit zusätzlichen Prozessoren, entsprechend den Array-Prozessoren bei üblichen Prozeßrechnern, ist prinzipiell möglich.

6 Zusammenfassung

Die Übersicht zeigt, daß momentan eine Vielzahl von Möglichkeiten bestehen, Algorithmen der digitalen Signalverarbeitung auf mehr oder weniger universellen, programmierbaren Systemen zu realisieren. Welche der Lösungen jeweils am sinnvollsten ist, hängt stark von den Randbedingungen ab und muß in jedem Fall getrennt entschieden werden. Hierbei ist zu bedenken, daß diese Entscheidung die Wirtschaftlichkeit neuer Produkte in starkem Maße beeinflußt. Es ist eine eindeutige Tendenz der Bauelementehersteller zu erkennen, diesen relativ jungen Bereich auch in der Zukunft durch die Entwicklung immer leistungsfähigerer Prozessoren und Spezialbausteine zu unterstützen. Optimistische Schätzungen sprechen von einer Erhöhung der Rechenleistung der integrierten Signalprozessoren um den Faktor 250 innerhalb der nächsten 10 Jahre [3]. Eine solche Entwicklung würde der digitalen Verarbeitung bislang nicht zugängliche Anwendungsbereiche im Bereich der Übertragungstechnik und der Signalverarbeitung bei Bandbreiten bis zu einigen MHz erschließen.

Für die Unterstützung der Arbeiten und die stetige Gesprächsbereitschaft möchte der Autor Herrn *Prof. Dr.-Ing. H. W. Schüßler* seinen besonderen Dank aussprechen.

Tabelle 6. Leistungsmerkmale MSP82

Registeroperationszeit	200 ns
Speicherzugriff	200 oder 400 ns
Rechenleistung bei Vektoroperationen	1,67...5 MOp/s
FFT 512 reelle Punkte	6 ms
Skalarprodukt Vektorlänge 100	40 µs
Matrixmultiplikation (20x20 Matrix)	16 ms
Polyphasenfilterbank:	
Abtastfrequenz am Eingang	8 kHz
Abtastfrequenz am Einzelkanalausgang	200 Hz
Länge der Filterimpulsantwort	1024
Anzahl der Kanäle von 0...4 kHz	128

Literatur

Allgemeines zur Signalverarbeitung

[1] *Schüßler, H. W.*: Digitale Systeme zur Signalverarbeitung. Springer-Verlag, Berlin, Heidelberg, New York, 1974.

[2] *Oppenheim, A. V., Schafer, R. W.*: Digital Signal Processing. Prentice-Hall, 1975.

Allgemeines zu Signalprozessoren

[3] *Broderson, R.*: VLSI for Signal Processing. Trends & Perspective in Signal Processing, Vol. 1 (1981), S. 7...11.

[4] *Kolb, H.-J.*: Mikroprozessoren in der digitalen Signalverarbeitung. Darmstädter Kolloquium „Mikroprozessoren und ihre Anwendung". R. Oldenbourg Verlag, 1979, S. 335...347.

[5] *Kolb, H.-J., Schüßler, H. W.*: Digitale Signalprozessoren. 4. Aachener Kolloquium Theorie und Anwendung der Signalverarbeitung, 1981, S. 17...23.

[6] *Salazar, A. C. (ed.)*: Digital Signal Computers & Processors. IEEE Press Selected Reprint Series 1977.

[7] *Brafman, J. P., Szczupak, J., Mitra, S. K.*: An Approach to the Implementation of Digital Filters using Microprocessors. IEEE Trans. on Acoustics, Speech, and Signal Processing. Vol. ASSP-26 (1978), S. 442...446.

[8] *Rackay, J. D., Silverman, H. F.*: A 16 bit Microprocessor-Based Digital Filter Architecture. IEEE Proc. ICASSP, 1978, S. 812...815.

[9] *Tan, B. S., Hawkins, G. J.*: Speed-optimized microprocessor implementation of a digital filter. Proc. IEE, vol. 128, Pt. E. (1981), S. 85...93.

[10] *Kolb, H.-J.*: A Signalprocessor using Bit Slice Elements for the Audio-Frequency Range. Signal Processing, vol. 2 (1980), S. 339...346.

[11] *Kolb, H.-J.*: Effective Programming and Realization of Real Time Signal Processors. Proc. of EUSIPCO 1980, Lausanne, 1980, S. 359...362.

Signalprozessor-Chips

29500 (AMD)

[12] *New, B. J.*: The Am29500 Signal Processing Family. IEEE Proc. Int. Conf. Acoustics, Speech and Signal Processing 1981, S. 378...381.

2811 (AMI)

[13] *Blasco, R. W.*: V-MOS chip joins microprocessor to handle signals in real time. Electronics, Aug. 30, 1979.

[14] S2816, Echo Cancellor Processor. Datenblatt der Firma AMI. September 1981.

[15] S2815, Digital Filter/Utility Peripheral. Datenblatt der Firma AMI.

[16] S2814, Fast Fourier Transformer. Datenblatt der Firma AMI. Januar 1981.

[17] S2811, Signal Processing Peripheral. Datenblatt der Firma AMI. Mai 1979

[18] *Nicholson, W. E., Blasco, R. W., Reddy, K. R.*: Schnelles Rechenwerk erweitert Mikroprozessor-Systeme. ELEKTRONIK 1979, H. 4, S. 53...60.

DSP (Bell)

[19] *Boddie, J. R., Daryanani, G. T., Eldumiati, I. I., Gadenz, R. N., Thompson, J. S., Walters, S. M.*: A Digital Signal Processor for Telecommunications Applications. Proc. ISSCC 1980, pp 44...45.

[20] Verschiedene Autoren und Beiträge: Digital Signal Processor. The Bell System Technical Journal, vol. 60, sept. 1981, pp 1431...1709.

2920 (Intel)

[21] The 2920 Analog Signal Processor Design Handbook. Intel August 1980.

[22] *Hoff, M. E., Townsend, M.*: Einchip-Signalprozessor verarbeitet Analogsignale in Echtzeit. ELEKTRONIK 1979, H. 14, S. 37...42.

[23] *Fliege, N.*: Digitale Filter mit dem Signalprozessor 2920. ELEKTRONIK 1981, H. 3, S. 81...85, H. 4, S. 89...94.

[24] *Müller, K. H.*: Echtzeit-Simulation mit dem Analog-Prozessor 2920. ELEKTRONIK 1981, H. 7, S. 95...98.

7720 (NEC)

[25] *Zoicas, A.*: Digital Filters, Application notes NEC, 1982.

[26] *Nishitani, T., Kawakami, Y. a. o.*: LSI signal processor development for communications equipment. IEEE Trans. Acoustics, Speech and Signal Processing, Vol. 3, pp 386...389, April 1980.

[27] *Kawakani, Y., Tanaka, H.*: 7720 User's manual. NEC-Japan 1980.

[28] *Epstein, D.*: 7720 Signal Processing Interface-Product description. NEC Microcomputers Inc. MA 1981.

[29] *Feldmann, J. A., Hofstetter, E. M.*: A Compact, Flexible LPC Vocoder based on a Commercial Signal Processing Microcomputer. IEEE Proc. Int. Conf. ASSP 1982.

[30] *Feldman, J. A.*: A Compact Digital Channel Vocoder using Commercial Devices. IEEE Proc. Int. Conf. ASSP 1982.

[31] *Nishitani, T., Aikoh, S., Araseki, T., Ozawa, K., Maruta, R.*: A 32 kb/s Toll Quality ADPCM Codec using a Single Chip Signal Processor. IEEE Proc. Int. Conf ASSP 1982.

(TRW)

[32] *Schirm, L.*: A Family of High Speed, Floating Point Chips. IEEE Proc. Int. Conf. Acoustics, Speech and Signal Processing 1981, S. 374...377.

32-Bit-Mikrocomputer für Signalverarbeitung und Prozeßsteuerung

Signalverarbeitung und Prozeßsteuerung erfordern komplexe Rechenvorgänge, die in kurzer Zeit ausgeführt werden müssen. Herkömmliche Prozessor-Architekturen sind diesen hohen Anforderungen nicht gewachsen, andere Konzepte sind notwendig. Bisherige Lösungen verwendeten Logikbausteine niedriger Integrationsdichte. Allerdings bedeutet das hohen Aufwand und lange Entwicklungszeit. Monolithische Prozessoren, die sich für die schnelle Signalverarbeitung eignen, wurden erst in der letzten Zeit entwickelt. Ein Beispiel dafür ist der in diesem Beitrag vorgestellte 32-Bit-Typ TMS320.

Der 32-Bit-Hochleistungs-Mikrocomputer TMS320 von Texas Instruments eignet sich insbesondere zur Implementierung komplexer digitaler Signalverarbeitungssysteme. Der Instruktionssatz ist so ausgelegt, daß dieser Prozessor auch in vielen Anwendungen der Datenverarbeitung und Steuerungstechnik eingesetzt werden kann. Wichtigste Merkmale dieses Einchip-Computers sind: 200 ns Instruktionszyklus, 32-Bit-Arithmetikeinheit, 16 × 16-Bit-Parallel-Multiplizierer (benötigt für die Operation lediglich einen Zyklus), 0...16-Bit-Barrel-Shifter, 288 Byte Daten-RAM (144 × 16), Interrupt mit vollem Kontext-Schutz, 3 KByte Programm-ROM (1,5 K × 16) auf dem Chip, die extern auf 8 K erweiterbar sind. Verschiedene Speicherbetriebsarten erleichtern anwendungsspezifische Lösungen.

Architektur

Die Blockschaltung des TMS320 ist in *Bild 1* dargestellt. Der Prozessor besitzt eine Architektur vom Harvard-Typ mit einem Akkumulator, bei der es möglich ist, Instruktionen zeitlich überlappend aufzunehmen und auszuführen. Möglich ist das, weil sich Daten und Programm in verschiedenen Speicheradreßbereichen befinden. Allerdings wurden Instruktionen vorgesehen, durch die die Bereiche miteinander zu verknüpfen sind. Aus diesem Grund kann man Konstanten für den Prozeß auch im eigentlichen Programmspeicher ablegen.

Herzstück des Prozessors ist eine schnelle Arithmetikeinheit, die aus einer 32-Bit-ALU, einem 16-Bit-Shifter, dem 16 × 16-Bit-Parallelmultiplizierer sowie einem 32-Bit-Akkumulator besteht. Daten werden entweder aus dem RAM über den Shifter oder aus dem Produkt-Register P in die ALU geladen. Daten aus dem RAM mit einer Wortbreite von 16 Bit werden nach Schiebeoperationen mit Vorzeichenzusatz für Zweier-Komplement-Arithmetik auf 32 Bit erweitert. Arithmetik-Operationen mit 32-Bit-Daten aus dem RAM werden auch durch eine Kombination spezieller Befehle, bei denen die Vorzeichenerweiterung im Shifter unterdrückt wird, unterstützt. Die ALU verfügt auch über logische Operationen für Steueranwendungen. Eine Sättigungs-Überlauf-Betriebsart dient zur Simulation von Sättigungsereignissen in Signalverarbeitungssystemen. Der 32-Bit-Akkumulator wird im Multiplex-Betrieb auf den 16-Bit-Datenbus umgeschaltet, damit die Resultate im RAM gespeichert werden können. Die höherwertigen Akkumulatorwerte können auch mit einem vorgegebenen Offset gespeichert werden, um die Skalierung der Resultate zu ermöglichen. *Bild 2a* zeigt ein Beispiel.

Adressierungsarten

Der Prozessor unterstützt vier Adressierungsarten. Die erste ist die direkte Adressierung, sie erfolgt von einem 7-Bit-Feld der Instruktion und einem Page-Register. Der Speicher ist für die direkte Adressierung in 128 Wort-Seiten aufgeteilt. Die zweite ist die indirekte Adressierung, bei der eines der zwei Hilfsregister (AR) Verwendung findet. Diese Register unterstützen die automatischen Inkrement-/Dekrement-Operationen parallel zu Speicher-Bezügen und Arithmetikoperationen. Daraus ergeben sich zwei verschiedene Adressierarten. Die Auswahl des AR-Registers als Quelle für eine Adresse wird durch das ARP-Register festgelegt. Der Datenspeicher ist so aufgebaut, daß ein Wort auf die nächsthöhere Adresse vom derzeitigen Speicherplatz in einem Maschinenzyklus dupliziert werden kann, während andere Operationen parallel ablaufen. In *Bild 2b* ist eine Instruktion gezeigt, die diese Eigenschaft besitzt, mit der Faltungen vereinfacht werden. Mit der Maschine sind außerdem einige direkte Operationen möglich, bei denen Teile des Instruktionsworts als Daten Verwendung finden. Aus diesem Grund können Programmkon-

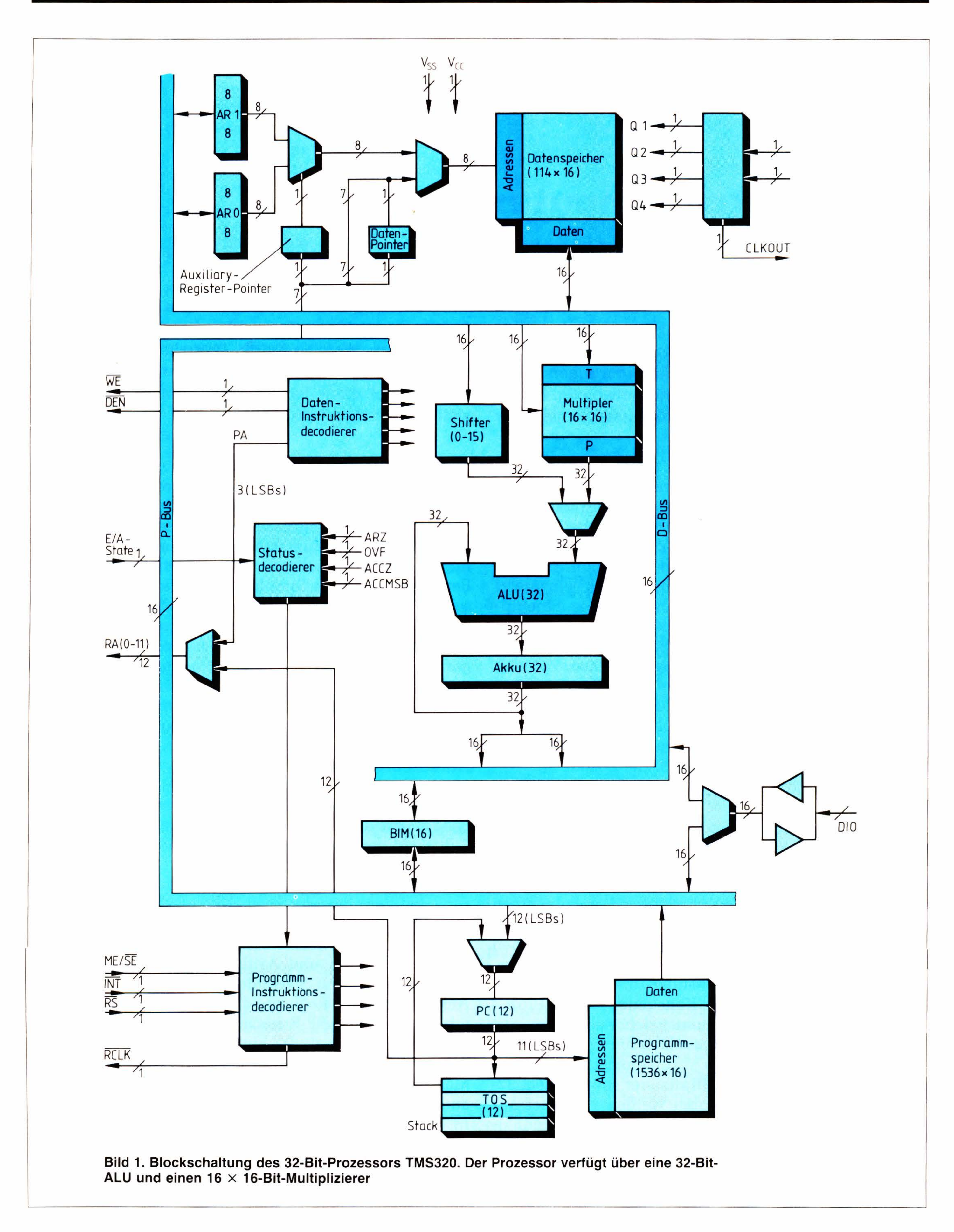


Bild 1. Blockschaltung des 32-Bit-Prozessors TMS320. Der Prozessor verfügt über eine 32-Bit-ALU und einen 16 × 16-Bit-Multiplizierer

stanten wie z. B. Filter-Koeffizienten Teil des Programms sein.

Auf dem Chip befindet sich ein Programm-ROM mit einer Organisation von 1536 Worten zu 16 Bit. Zur Erhöhung des Durchsatzes sind Programm-Steuerfunktionen von Daten-Manipulationen getrennt. Daher verfügt der Programm-Zähler über seinen eigenen Inkrementer und einen 4-Ebenen-Stack für Unterprogramm-Steuerung.

Software

Der Prozessor verwendet 16- und 32-Bit-Instruktionen. Bei Interrupts und Unterprogramm-Aufrufen wird der volle Maschinen-Kontext gesichert. Verzweigungen können bei den meisten arithmetischen Bedingungen, bei Überlauf, Register 0 und ohne Bedingung erfolgen. Es ist ebenfalls möglich, den Akkumulator-Inhalt als Programm-ROM-Adresse zu verwenden, um an Konstanten, die im ROM gespeichert sind, zu gelangen, und zum Unterprogrammaufruf bei datenabhängigen Verarbeitungsvorgängen. Ein Beispiel für eine solche Instruktion zeigt *Bild 2c*. Der Prozessor ist zur Unterstützung von zwei Arten der Programm-Speicher-Operation konfiguriert. Die erste Konfiguration (TMS320M10) besitzt ein Programm-ROM auf dem Chip und eignet sich daher für Einchip-Anwendungen mit hohen Stückzahlen. Eine Kombination aus internem ROM (1,5 K Worte) und externem Programmspeicher (2,5 K Worte) ist ebenfalls möglich und kann z. B. bei Systemen mit festen Kern-Routinen und unterschiedlichen Anwendungs-System-Konfigurationen verwendet werden. Die zweite Ausführung (TMS32010) unterstützt 4096 Worte externen Programmspeicher und unterdrückt das interne ROM. Diese Ausführung ermöglicht, daß das Benutzer-Programm in einem externen 4-K-Speicher untergebracht ist und die Entwicklungssystem-Software in dem Programm-ROM auf dem Chip. Daher kann dieses Bauelement nicht nur zur Unterstützung der Eigen-Emulation, sondern auch als das eigene Entwicklungssystem verwendet werden. Das Interface des externen Programmspeichers arbeitet mit der gleichen Geschwindigkeit wie das interne ROM, wodurch Echtzeit-Entwicklung und -Ausführung von Benutzer-Programmen möglich ist.

Ein-/Ausgangs-Operationen werden über einen parallelen 16-Bit-Bus ausgeführt, auf dem acht Kanäle definiert sind. Der Prozessor kann E/A-Operationen mit

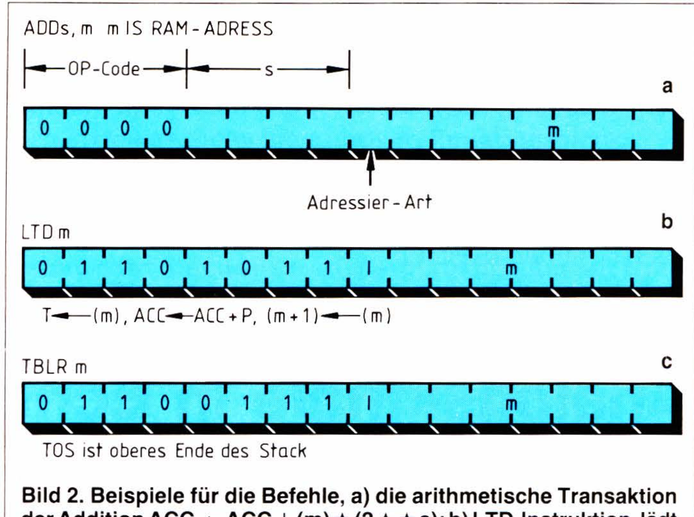


Bild 2. Beispiele für die Befehle, a) die arithmetische Transaktion der Addition ACC ← ACC + (m) ★ (2 ★ ★ s); b) LTD-Instruktion, lädt das T-Register, berechnet Produkt, dupliziert Daten im RAM T ← (m), ACC ← ACC + P, (m + 1) ← (m); c) TBLR-Instruktion (Table Read From ROM) TOS ← PC, PC ← ACC, (m) ← (PC), PC ← TOS

einer Rate von 40 Mio. Bit/s ausführen. Ein Polling-Eingang ermöglicht softwaremäßige E/A-Steuerung, außerdem ist ein Interrupt-Anschluß für Hardware-E/A und Multitasking vorgesehen.

Der Chip

Hergestellt wird der 32-Bit-Prozessor TMS320 mit einem 3-µm-Silizium-Gate-NMOS-Prozeß. Er umfaßt eine Chipfläche von 43,81 mm^2. Die Bausteine sind in 40poligen DIL-Gehäusen untergebracht und weisen eine Verlustleistung von 950 mW auf. Bei einer maximalen Taktfrequenz von 20 MHz beträgt die Befehls-Rate 5 Mio. Instruktionen/s. Verschiedene Test-Arten sind zur Produktions-Überwachung und Ausbeute-Analyse vorgesehen. Für Prüfzwecke und Zuverlässigkeits-Analysen sind Prozessor, Datenspeicher und PROM auf dem Chip einzeln ansprechbar.

Anwendungen

Die Kombination von hohem Durchsatz, 32-Bit-Operationen, leistungsfähigem Mikroprozessor-Befehlssatz und umfangreichem Programmspeicher auf dem Chip eröffnen neue Möglichkeiten für Anwendungen in der digitalen Signalverarbeitung und Steuerungstechnik. Anwendungsgebiete für den TMS320 liegen daher in den Bereichen Sprachsynthese/-analyse/-erkennung, Bild-Verarbeitung, Modems, schnelle Steuerungen, Verarbeitung numerischer Daten.

Wichtigste Merkmale des 32-Bit-Prozessors TMS320:

Befehlszyklus 200 ns
16-Bit-Befehls-/Daten-Worte
1,5-K-ROM
144 Worte RAM
16 × 16-Bit-Hardware-Multiplikation (1 Zyklus)
32-Bit-ALU und -Akkumulator
Selbst-Emulation
Eine Betriebsspannung (+5 V)

Nach Unterlagen der Fa. Texas Instruments, bearbeitet von *P. von Bechen*.

M. E. Hoff
Matt Townsend

Einchip-Signalprozessor verarbeitet Analogsignale in Echtzeit

Die digitale Echtzeitverarbeitung von Analogsignalen – attraktiv, weil Kurvenformen von einem Programm wie Zahlen behandelt werden – war bisher auf spezielle Hochgeschwindigkeitsprozessoren beschränkt. Nur sie konnten die nötigen Berechnungen schnell genug ausführen. Inzwischen bietet ein NMOS-Baustein beim Entwurf von Analogsystemen dieselben Möglichkeiten: der Echtzeit-Signalprozessor, der hier am Beispiel des Typs 2920 vorgestellt wird.

Der Baustein 2920 (Intel) entstand aus dem drängenden Verlangen nach einer programmierbaren Einheit, die die verschiedenen Prozeduren und Modulationstechniken der Nachrichtentechnik unterstützt. Er kombiniert AD- und DA-Umsetzer mit einem Spezial-Mikrocomputer zu einem Element mit einem einzigartigen Befehlssatz, der es gestattet, komplette Analog-Untersysteme zu programmieren. Die Funktionen, die mit Hilfe des Bausteins zu realisieren sind – Filtern, Modulieren, Demodulieren, Begrenzen, Mischen usw. –, erfordern normalerweise zahlreiche passive Bauelemente, Operationsverstärker und andere diskrete, analoge Elemente. Mit geringem externen Aufwand kann der 2920 als Modem, Entzerrer, Tonquelle oder Tonempfänger beschaltet werden. Er kann auch zur Prozeßsteuerung und als (Servo-)Motortreiber eingesetzt werden.

Die Programmierbarkeit ist der wesentliche Vorzug des Echtzeit-Signalprozessors: Seine endgültige Funktion hängt vom Inhalt des EPROMs ab, das sich mit auf dem Chip befindet.

1 Voraussetzungen für den Befehlssatz

Der Prozessor unterscheidet sich erheblich von konventionellen µPs. Es gibt aber Ähnlichkeiten: Er verarbeitet digitale Daten eines konventionellen AD-Umsetzers (am Eingang) und sendet das Ergebnis zu einem DA-Umsetzer (am Ausgang). Der Unterschied liegt in der Art der Berechnung, die für die Signalverarbeitung erforderlich ist. Außerdem muß der Prozessor bei jedem Wert vom AD-Umsetzer sein gesamtes Programm durchlaufen, da die Funktion auf einem Echtzeit-Abtastsystem basiert. Die Programmausführungszeit bestimmt somit die Abtastrate, und das hat Beschränkungen in mancher Hinsicht zur Folge.

Im allgemeinen ist eine feste Abtastrate notwendig, da die Charakteristik eines jeden simulierten Analogsystems stark von ihr beeinflußt wird. In der Tat können schon kleine Abweichungen ein nicht zu vernachlässigendes Rauschen bewirken. Deshalb muß die Zeit für einen Programmdurchlauf notgedrungen immer dieselbe sein. Um das zu erreichen, wird jeder Befehl in der gleichen Zeit ausgeführt. Außerdem enthält der Befehlssatz aus diesem Grund keine bedingten Sprünge (ausgenommen den Sprung vom Programm-

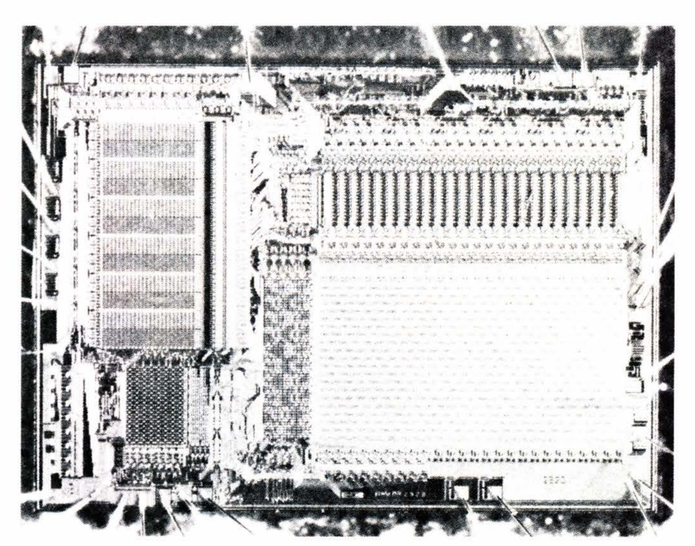


Bild 1. Komplette Analog-Untersysteme können mit dem Echtzeit-Signalprozessor 2920 (Intel) programmiert werden. Auf dem gut 30 mm² großen Chip sind eine spezielle CPU sowie AD- und DA-Umsetzer untergebracht, die von einem EPROM gesteuert werden

ende zurück zum Anfang). Bedingte Operatoren jedoch erlauben logische Entscheidungen, ohne die Ausführungszeit zu beeinflussen.

Obwohl diese Techniken das Programmieren in gewisser Hinsicht beschränken, sorgen sie für eine feste Abtastrate, die auf einem Programmdurchlauf basiert. Man kann diese Abtastrate somit einfach berechnen, indem man die Befehlsausführungsrate durch die Anzahl der Programmschritte teilt.

Für die digitale Signalverarbeitung nach dem Abtastverfahren muß der Mikroprozessor außerordentlich schnell sein, wenn eine annehmbare Bandbreite erzielt werden soll. Das Nyquist-Kriterium besagt: Ein kontinuierliches System kann mit dem Abtastverfahren exakt simuliert werden, wenn die Abtastrate mindestens doppelt so hoch ist wie die höchste vorkommende Frequenz. In der Praxis sind aber meistens noch höhere Abtastraten erforderlich. Überdies muß der Prozessor extrem schnell rechnen, da alle Operationen zur Erzeugung eines einzigen Ausgangswertes für jeden Abtastwert auszuführen sind. Schon eine Bandbreite von 10 kHz würde beispielsweise eine Abtastrate von 20 kHz bedingen, was einem kompletten Programmdurchlauf alle 50 µs entspricht. Nimmt man an, daß 100 Befehle ausgeführt werden, dann verbleiben pro Befehl noch 500 ns – eine Zeit, die nur wenige Minicomputer schaffen. Um so erstaunlicher ist es, daß sie ein Baustein von gut 30 mm² Chipfläche (*Bild 1*) schafft, der noch dazu mit einem Standard-N-Kanal-MOS-Prozeß hergestellt wird.

2 Funktionseinheiten

Wie aus *Bild 2* hervorgeht, teilt sich der 2920 in drei Funktionseinheiten auf: Programmspeicher, Arithmetikteil und Analogteil. Der EPROM-Programmspeicher steuert die beiden übrigen Funktionsblöcke.

Vier Analogeingänge und acht Analogausgänge hat der Baustein. Ein Multiplexer sorgt dafür, daß nur ein Sample-and-hold-Verstärker nötig ist. Die AD-Umsetzung wird durch sukzessive Approximation mit Hilfe des Ausgangs-DA-Umsetzers (arbeitet mit Widerstandsnetzwerk) durchgeführt. Ein Demultiplexer liefert die Signale für die acht gepufferten Ausgänge, von denen jeder seinen eigenen Sample-and-Hold-Verstärker hat. Das Datenregister stellt die Verbindung zum digitalen Teil des Chips dar.

Der Mikroprozessor im 2920 enthält ein Scratchpad-RAM mit zwei Ports, einen Binärteiler und eine arithmetisch/logische Einheit (ALU). Das Datenregister, das vom Analogteil benutzt wird, ist in Wirklichkeit ebenfalls Teil des RAMs, so daß es der Prozessor als adressierbaren Speicher ansieht.

Obwohl AD- und DA-Umsetzer auf 9 Bit genau sind, arbeiten sie intern mit 25 Bit. Um insgesamt einen geringen Rundungsfehler zu erreichen, ist nämlich für Zwischenrechnungen eine wesentlich höhere Genauigkeit erforderlich, als für das Endergebnis verlangt wird. Sollte ein Überlauf auftreten, geht der Prozessor „in die Sättigung". Mit anderen Worten, er ersetzt das Ergebnis sofort durch den größten speicherbaren Wert.

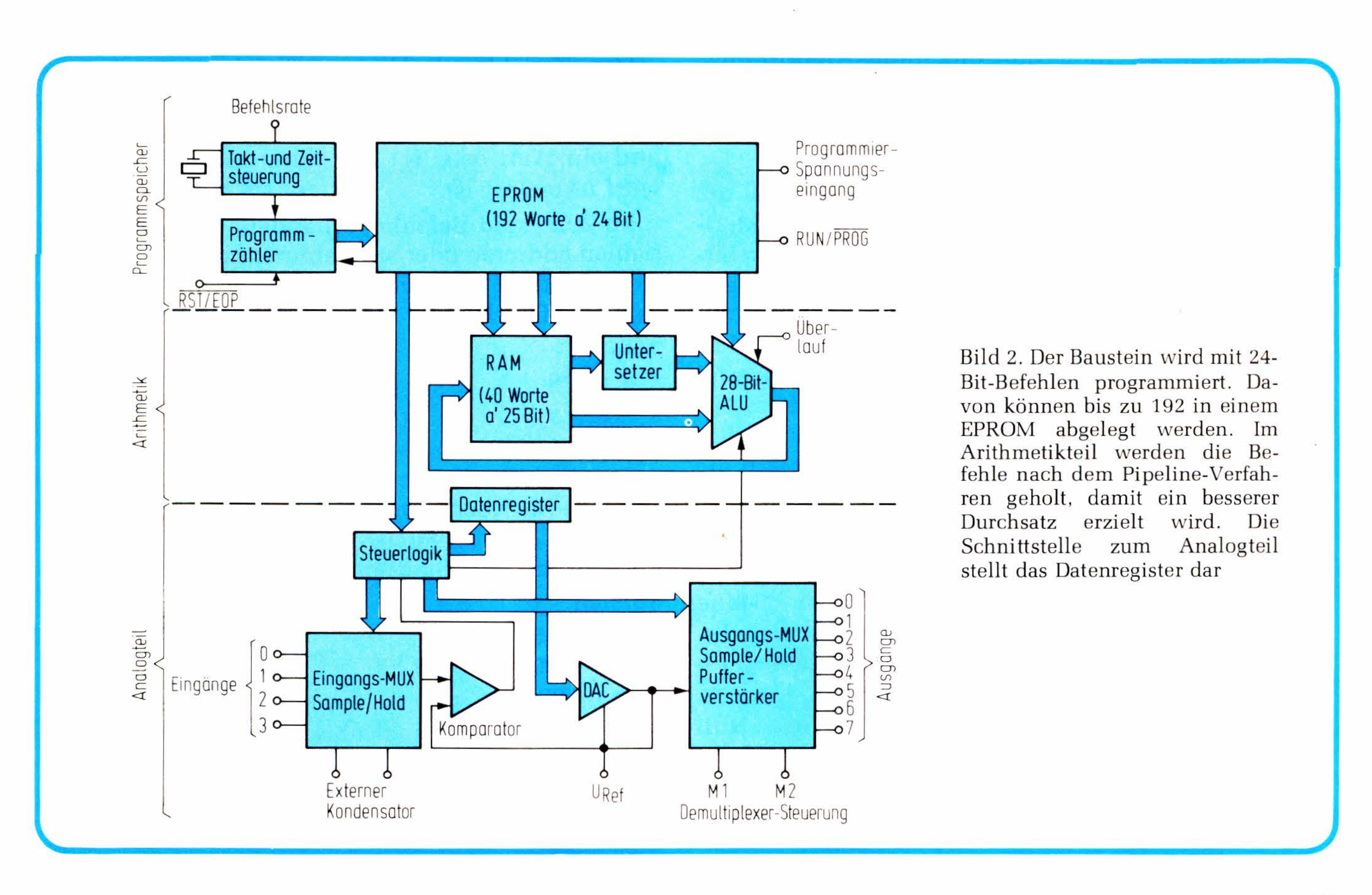


Bild 2. Der Baustein wird mit 24-Bit-Befehlen programmiert. Davon können bis zu 192 in einem EPROM abgelegt werden. Im Arithmetikteil werden die Befehle nach dem Pipeline-Verfahren geholt, damit ein besserer Durchsatz erzielt wird. Die Schnittstelle zum Analogteil stellt das Datenregister dar

Jeder der 40 RAM-Speicherplätze ist 25 Bit „breit". Adressiert werden sie mit einem 6-Bit-Wort; die verbleibenden 24 Adressen wählen vorbestimmte Konstanten und das Datenregister aus. Um den Durchsatz zu erhöhen, wurde das RAM mit Zweifach-Port-Zellen versehen, die über jeden der Ports adressiert werden können.

Das EPROM ist in der Lage, bis zu 192 Befehle mit je 24 Bit zu speichern. Das Befehlsformat besteht aus fünf aufeinanderfolgenden Feldern: Digital-Operator, Quelladresse, Zieladresse, Shift-Maß (*extent of shifting*) und Analog-Operator. Ein derart langes Wort ist vergleichbar mit einem Mikroprogramm-Wort in Computern mit Steuerspeicher – es führt mehrere Operationen auf einmal aus. In diesem Fall handelt es sich um Speicher-zu-Speicher-Operationen.

Bei maximaler Geschwindigkeit führt der Signalprozessor jeden Befehl in 400 ns aus. Das längste Programm (192 Befehle) dauert dann 76,8 µs, was einer Abtastrate von etwa 13 kHz entspricht. Nach Nyquist ist die Bandbreite dieses Systems 6,5 kHz. Kürzere Programme ermöglichen natürlich höhere Bandbreiten. Bestimmte Techniken – z. B. das mehrfache Ablegen eines Programms im Stack – können die Bandbreite ebenfalls erhöhen.

Um die Geschwindigkeit so hoch wie möglich zu treiben, wird im Baustein das sogenannte Pipeline-Verfahren angewendet. Dabei werden jeweils vier Befehle auf einmal vom EPROM geholt, und die Hol-Phase für die nächsten vier überlappt sich mit der Hol-Phase der vorhergehenden vier. Überdies führen alle arithmetischen Befehle das sogenannte „Carry-lookahead" über die gesamte Breite des Akkumulators durch.

Simulation mit Abtastsystemen

Die Funktion des Signalprozessors basiert auf der Abtasttheorie, die besagt, daß eine stetige Funktion von einer Folge periodischer Abtastwerte exakt dargestellt wird, vorausgesetzt die Abtastrate ist hoch genug. Aber gerade diese Forderung – und der enorme Rechenaufwand zwischen zwei Werten – haben die digitale Verarbeitung von Analogsignalen auf leistungsfähige größere Computer beschränkt. Der 2920 löst das Rechenproblem mit Hilfe der Pipeline-Struktur und einem ausgefeilten Multiplikations-Algorithmus, der einen wesentlich geringeren Schaltungsaufwand und weniger Schritte benötigt als der herkömmliche Shift/Addier-Algorithmus.

Das unten dargestellte aktive Filter (A) hat ein konjugiert-komplexes Polpaar. Seine charakteristischen Werte (Übertragungsfunktion, Verstärkung, Lage der Pole) sind durch die Gleichungen gegeben. Es kann durch das Abtastsystem B simuliert werden. Die Kreise mit einem x stellen Multiplizierer dar, die mit dem Summenzeichen sind Addierer. Durch die Blöcke mit z^{-1} werden Zeitverzögerungen von einer Abtastperiode wiedergegeben. Die Koeffizienten β_1 und β_2 steuern die Frequenzparameter des Filters, während mit G die Verstärkung eingestellt wird.

Die Gleichungen in B ergeben die simulierten charakteristischen Werte. Obwohl es sich um eine Simulation handelt, ergeben sich dieselben Ergebnisse wie in A, wenn unendlich hohe Frequenzen zugrunde gelegt werden: Für $\beta_1 = 2 \cdot e^{-aT}$ ergibt sich 2, wenn die Abtastperiode T gegen Null geht; und $\beta_2 = -e^{-2aT}$ wird –1 für T gegen Null.

Für endliche Frequenzen allerdings können sich erhebliche Abweichungen der Filter-Charakteristik ergeben, selbst wenn die Koeffizienten nur geringe Fehler aufweisen. Alle Berechnungen müssen deshalb mit hoher Genauigkeit durchgeführt werden.

Ein Computer kann das System in B mit folgenden drei Gleichungen simulieren:

$y_2 = y_1$
$y_1 = y_0$
$y_0 = \beta_1 \cdot y_1 + \beta_2 \cdot y_2 + G \cdot x$

Den Variablen links vom Gleichheitszeichen werden die Werte auf der rechten Seite zugewiesen. An jeder Multiplikation sind eine Variable und ein Wert beteiligt, der durch den Filterentwurf festgelegt ist.

Im 2920 sind Befehle vorhanden, die zwei Variablen addieren oder subtrahieren, wobei die erste mit einer Potenz von 2 skaliert wird. Ein einzelner Befehl könnte somit folgende Formen annehmen:

$x = y\ (2^k)$
$x = x + y\ (2^k)$
$x = x - y\ (2^k)$

Der Nutzen dieser Skalierung wird bald klar werden. Die Koeffizienten können nämlich durch Summen oder Differenzen von Zweierpotenzen ausgedrückt werden, wie in den folgenden Beispielen:

$\beta_1 = 1{,}7656 = 2^1 - 2^{-2} + 2^{-6}$
$\beta_2 = -0{,}99414 = -2^0 + 2^{-7} - 2^{-9}$
$G = 0{,}00293 = 2^{-8} - 2^{-10}$

Der Signalprozessor führt diese Rechnungen schnell und mit wenig Schaltungsaufwand aus. Die Filterstufe von B wird durch die 2920-Befehle

Die ALU ist für folgende grundlegenden Operationen ausgelegt: Datentransport, Addition, Subtraktion, Absolutwert und verschiedene logische Operationen. Jeder elementare Maschinenbefehl holt zwei Operanden vom Scratchpad-RAM, schickt den ersten durch den Binärteiler, führt die gewünschte arithmetische Funktion aus und ersetzt den zweiten Operanden durch das Ergebnis.

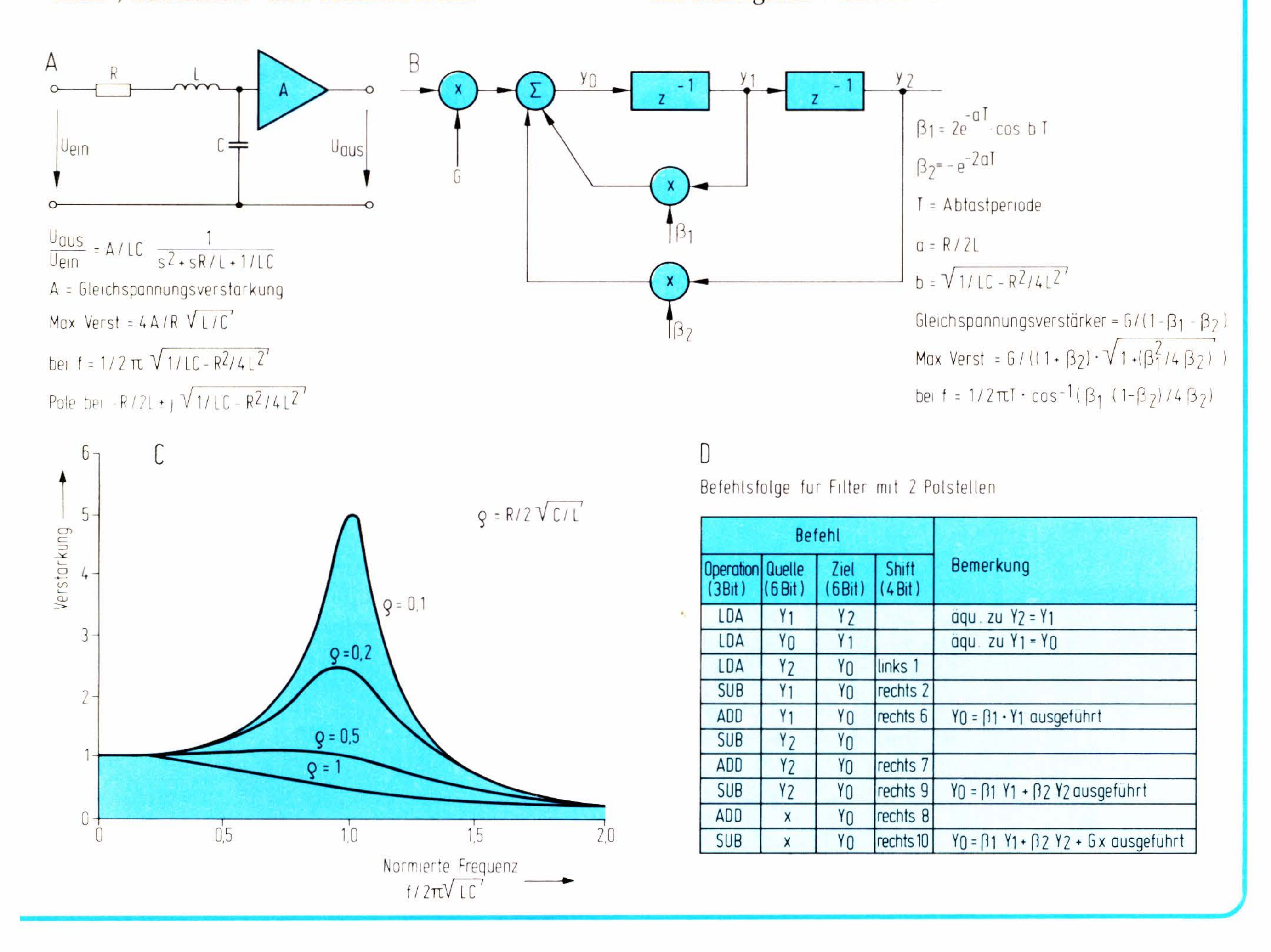
Schlüsselelemente für die hohe Geschwindigkeit und die hohe Genauigkeit der Arithmetik ist der Binärteiler. Im Gegensatz zu den herkömmlichen Shift/Addier-Multiplizierern, die einen Zyklus pro Bit benötigen, führt der 2920 Sequenzen von Additionen und Subtraktionen mit Werten durch, die vorher entsprechend geteilt worden sind (siehe Kasten S. 39). Bei der Multiplikation von Variablen mit Konstanten wird dadurch die Zahl der Zyklen auf etwa ein Drittel reduziert (da die meisten Filter in Analoganwendungen nicht variabel sind, wird gewöhnlich mit Konstanten multipliziert).

Einige der grundlegenden ALU-Operationen werden abhängig von einer Bedingung ausgeführt, sie benutzen bestimmte Bits des Datenregisters, die normalerweise der AD- und DA-Umsetzung vorbehalten sind. Multiplikation und Division von zwei Variablen werden beispielsweise durch bedingte Addition und Subtraktion möglich. Schließlich sind mit bedingten Operationen unstetige Übertragungsfunktionen realisierbar.

AD- und DA-Umsetzer haben eine Genauigkeit von 9 Bit, wenn man das zusätzliche Vorzeichenbit mitrechnet. *Bild 3* zeigt die Umschaltlogik am Eingang, die dafür sorgt, daß ein korrektes Vorzeichenbit hin-

in D unmittelbar realisiert. Ein Links-Shift um eine Stelle ist äquivalent zu einer Multiplikation mit 2^1, ein Rechts-Shift um sechs Stellen entspricht der Multiplikation mit 2^{-6} usw. LDA, ADD und SUB repräsentieren Operations-Codes für Lade-, Subtrahier- und Addierbefehl.

Diese Methode erlaubt viel schnellere Multiplikationen mit Konstanten als das konventionelle Shift/Addier-Verfahren. Das wirkt sich in dieser Anwendung besonders günstig aus, weil solche Multiplikationen in Digitalfilter-Berechnungen am häufigsten vorkommen.



Befehlsfolge fur Filter mit 2 Polstellen

Befehl				Bemerkung
Operation (3Bit)	Quelle (6 Bit)	Ziel (6Bit)	Shift (4 Bit)	
LDA	Y_1	Y_2		äqu. zu $Y_2 = Y_1$
LDA	Y_0	Y_1		äqu. zu $Y_1 = Y_0$
LDA	Y_2	Y_0	links 1	
SUB	Y_1	Y_0	rechts 2	
ADD	Y_1	Y_0	rechts 6	$Y_0 = \beta_1 \cdot Y_1$ ausgeführt
SUB	Y_2	Y_0		
ADD	Y_2	Y_0	rechts 7	
SUB	Y_2	Y_0	rechts 9	$Y_0 = \beta_1 Y_1 + \beta_2 Y_2$ ausgeführt
ADD	x	Y_0	rechts 8	
SUB	x	Y_0	rechts 10	$Y_0 = \beta_1 Y_1 + \beta_2 Y_2 + Gx$ ausgeführt

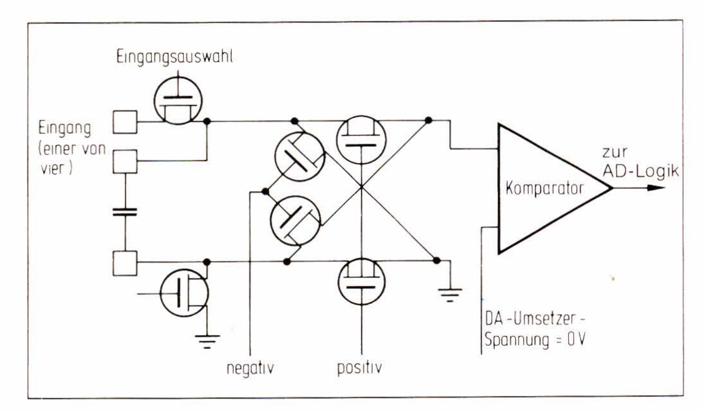


Bild 3. Mit einem zusätzlichen Vorzeichenbit zu den vorhandenen 8 Bit wird die Genauigkeit auf 9 Bit gesteigert. Der Baustein hat am Eingang eine Umschaltlogik, die dafür sorgt, daß ein Signal korrekter Polarität vom Sample-and-Hold-Kondensator zum Komparator der AD-Logik geliefert wird

zugefügt wird, während nur eine einzige (positive) Referenzspannung benötigt wird.

Der DA-Umsetzer ist um eine Schaltermatrix und eine Widerstandskette herum aufgebaut, die in sich mehrfach „gefaltet" ist (*Bild 4*). Diese Anordnung verringert die Empfindlichkeit gegenüber Temperaturschwankungen und Unregelmäßigkeiten in der Chip-Oberfläche. Zusammen mit den Tatsachen, daß der Baustein quarzgesteuert ist, daß eine externe Referenzspannung benutzt wird und daß alle Berechnungen digital ausgeführt werden, ergibt das analoge Untersysteme, die wesentlich stabiler sind als ihre vollanalogen Gegenstücke.

Der Signalprozessor ist in der Lage, Filter mit bis zu 40 Polstellen oder 20 konjugiert-komplexen Polpaaren zu ersetzen. Das reicht für viele komplexe Analogsysteme aus, z. B. für „Dual-tone-Multifrequency"-(DTMF) oder „Touch-tone"-Empfänger, wie sie in Telefonen verwendet werden.

3 Anwendungsbeispiel

Die Möglichkeiten des Bausteins werden eindrucksvoll demonstriert von einem Tonfrequenz-Analysator, der das Übertragungsverhalten des Prüflings auf dem Oszilloskop anzeigt (*Bild 5*). Er liefert ein Vertikal- und ein Horizontalsignal, die direkt zum Oszilloskop geführt werden, und mischt das Eingangssignal mit dem Ausgang eines Wobbel-Oszillators.

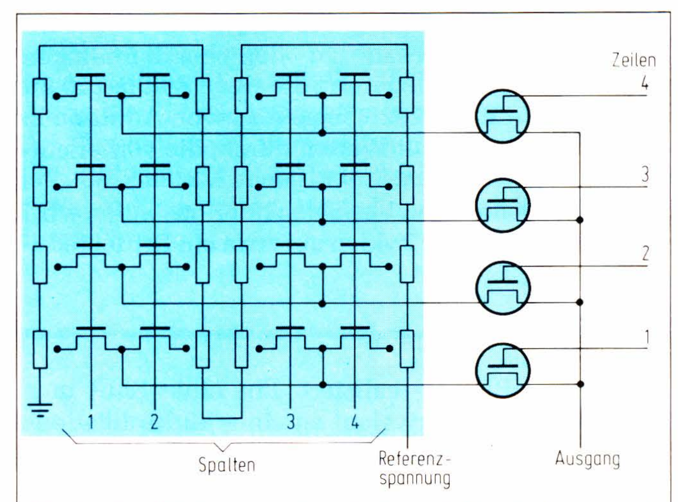


Bild 4. Der DA-Umsetzer ist mit einem Widerstandsnetzwerk aufgebaut, das durch mehrfaches Falten quadratisch angeordnet ist. Dadurch vermindern sich die Einflüsse von Temperatur und unterschiedlicher Oberflächenbeschaffenheit

Das zu analysierende Eingangssignal wird zuerst mit einem einfachen externen Netzwerk gefiltert, das hochfrequente Anteile unterdrückt. Im 2920 passiert es dann zunächst das Äquivalent eines Tiefpaßfilters zur weiteren Bandbegrenzung. Dann wird es – vor dem Mischen – durch automatische Verstärkungsregelung moduliert. Man erreicht das mit einem einfachen Divisions-Algorithmus, dessen Divisor gewon-

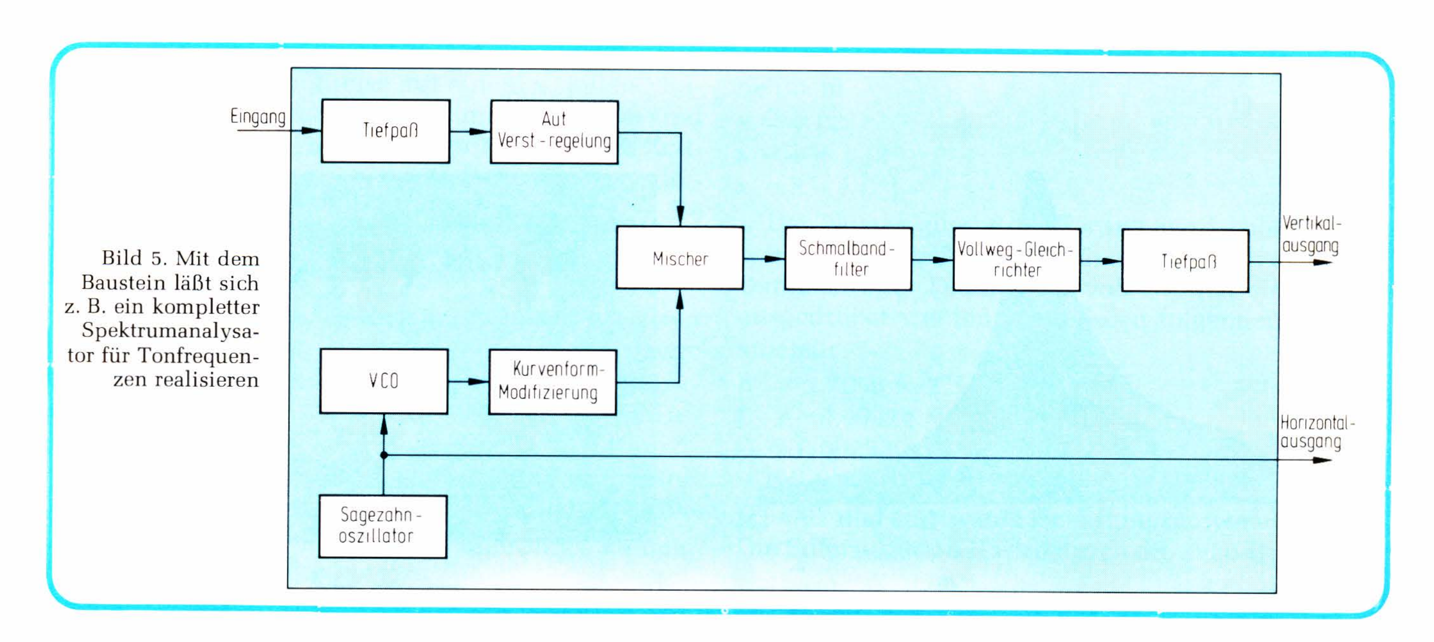


Bild 5. Mit dem Baustein läßt sich z. B. ein kompletter Spektrumanalysator für Tonfrequenzen realisieren

Tabelle. Speicherplatzbedarf für Spektrumanalysator

Funktionsblock	ROM-Worte	RAM-Worte
Tiefpaß (Eing.)	20	5
Aut. Verst.-Regelung	18	1
Wobbel-Oszillator	5	1
VCO	7	1
Kurvenform-Mod.	10	1
Mischer	12	0
Schmalbandfilter	30	6
Gleichrichter	1	0
Tiefpaß (Ausg.)	10	2
Gesamt	113	17

nen wird, indem man den Absolutwert des Signals durch einen Tiefpaß schickt (um den Mittelwert zu bekommen).

Ein zweiter Programmteil simuliert ein Oszillatorpaar, von dem der eine die Spektrum-Abtastrate und damit die Wobbelfrequenz (Horizontaleingang des Oszilloskops) bestimmt, während der zweite einen VCO darstellt, der vom ersten gesteuert wird. Die VCO-Frequenz, die den interessierenden Bereich überstreicht, wird schließlich mit dem Eingangssignal gemischt.

Beide Oszillatoren erzeugen lineare Sägezähne. Da das entsprechende Programm aber nur diskrete Werte für die simulierten Oszillatoren berechnet, könnten geringe Abweichungen zu Problemen führen, wenn die Harmonischen des Sägezahns mit der Abtastfrequenz ungewollte Mischprodukte bilden. Aus diesem Grund wird das Ausgangssignal des zweiten Oszillators einer nichtlinearen Transformation unterzogen, die es annähernd sinusförmig macht. Der Prozessor führt diese Transformation mit einer stückweise linearen Approximation durch, die Absolutwert-Funktionen mit den Effekten der erwähnten Überlauf-Sättigung der ALU kombiniert. Der erste Oszillator benötigt eine derartige Transformation nicht, da er so niedrige Frequenzen erzeugt, daß Mischprodukte mit der Abtastfrequenz auf jeden Fall unbedeutend sind.

Der Mischer wird mit einer Multiplikationsroutine simuliert, die das Ausgangssignal des zweiten Oszillators mit dem gefilterten und amplitudengeregelten Nutzsignal mischt. An seinem Ausgang erscheinen Summen- und Differenzfrequenz. Da nur das Summensignal von Interesse ist, wird es mit einem Schmalbandfilter isoliert. Seine Amplitude entspricht genau der des Eingangssignals bei einer Frequenz, die von der Differenz zwischen Mittenfrequenz des Schmalbandfilters und VCO-Frequenz bestimmt wird. Der Absolutwert-Algorithmus des Prozessors führt eine Vollweggleichrichtung nach dem Schmalbandfilter durch. Danach folgt noch ein Tiefpaß, und das Signal wird über den Vertikalausgang ausgegeben. Da Horizontal- und Vertikalausgang des Bausteins nach Sample-and-Hold-Schaltungen liegen, ist es empfehlenswert, daß man zur Glättung noch einfache Tiefpässe einsetzt.

Die im Spektrumanalysator verwendeten Filter haben einfache oder konjugiert-komplexe Polstellen. Filter mit mehreren Polstellen werden einfach aus diesen Grundelementen zusammengesetzt. Der Signalprozessor kann auch Filter mit endlicher Impulsantwort simulieren, und er kann Nullstellen entweder unabhängig oder zwischen Polstellen einfügen.

Der Analysator überstreicht einen Frequenzbereich von 300 Hz...3 kHz zehnmal pro Sekunde mit einer Auflösung von etwa 100 Hz. Das Schmalbandfilter hat eine Mittenfrequenz von 4,5 kHz und die Abtastrate beträgt ungefähr 13 kHz.

In der *Tabelle* ist der jeweils benötigte Platzbedarf im EPROM und im RAM für die verschiedenen Funktionsblöcke angegeben. Die Angaben beziehen sich auf die beschriebenen Parameter und können für andere Werte abweichen. Da das Programm nicht den gesamten Speicher belegt, könnten mehr Funktionen implementiert werden, oder die Abtastrate könnte auf 20 kHz erhöht werden. Eine weitere Möglichkeit ist, die Abtast- und Filter-Algorithmen für das Eingangssignal mehrfach zu programmieren, um die effektive Abtastrate zu erhöhen. Damit würde ein einfacheres externes Filter ausreichen.

4 Unterstützung

Die Produktion des 2920 soll Ende des Jahres anlaufen. Da es sich um einen Mikroprozessor mit einigen Zusätzen handelt, muß er zumindest ebenso gut unterstützt werden wie gegenwärtige µPs. Intel plant einen Assembler und einen Simulator, die beide (resident) auf dem Entwicklungssystem Intellec laufen. Da die Arithmetik im Zusammenhang mit dem Entwurf und der Optimierung von digitalen Filtern außerordentlich komplex ist, will man auch ein Software-Paket herausbringen, das diese Tätigkeiten unterstützt und den Assembler-Code für den 2920 generiert. Diese Hilfsmittel werden den Entwickler von Analogsystemen so unterstützen, wie es der µP-Entwickler inzwischen gewöhnt ist.

Autorisierte Übersetzung und genehmigter Nachdruck aus Electronics Bd. 52, H. 5, copyright McGraw-Hill, Inc. 1979. Aus dem Englischen übertragen von Rudolf Hofer.

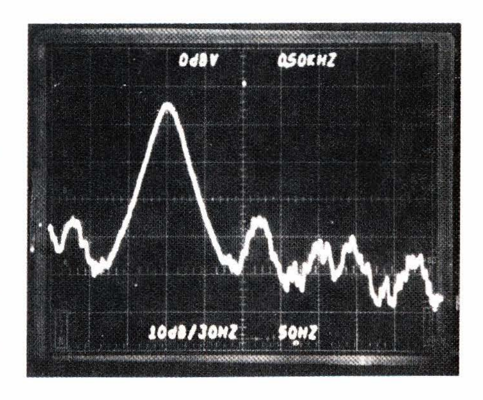


Bild 6. Spektrum eines einfachen Filters 2. Ordnung mit 400 Hz Mittenfrequenz: Das entsprechende Programm belegt 10 Worte im EPROM – 182 Worte bleiben frei für weitere Anwendungen

David Quong, Robert Perlman

FFT-Adressier-IC zur Transformation von über 65 000 Punkten

Schnelle Fourier-Transformation (FFT) zählt zu den nützlichsten Algorithmen im Bereich der digitalen Signalverarbeitung. Sie stellt ein schnelles und bequemes Mittel zur Berechnung des Frequenzspektrums eines Signals dar. In Kombination mit anderen Operationen lassen sich außerdem schnelle Korrelationen und Konvolutionen ausführen. FFT findet man in Anwendungen wie z. B. Radar, Echolot, Bildverarbeitung und Spektralanalyse. Der programmierbare FFT-Adreßsequenzer Am29540 vereinfacht den Hardware-Aufbau für ein FFT-System, indem er alle Daten und Koeffizientenadressen, die zur Ausführung der FFT erforderlich sind, erzeugt. Aufgrund der Programmierbarkeit des Bauelementes lassen sich die Adressen für die Fourier-Transformationen im Bereich von 2...65 536 Punkten erzeugen.

Das Problem der Adreßerzeugung

Eines der schwierigsten Probleme, das beim Entwurf von FFT-Hardware zu lösen ist, ist die Adressierung der Daten- und Koeffizientenspeicher. Ein kurzer Blick auf die Struktur von FFT-Hardware zeigt, warum das so ist:

Herzstück des FFT-Algorithmus ist die Butterfly-Operation, die ihren Namen aufgrund ihrer schematischen Darstellung trägt. *Bild 1a* zeigt die Butterfly-Operation für eine Wurzel-2-DIT-FFT (Decimation-in-Time). Im „Schmetterling" gibt es die zwei komplexen Datenpunkte A und B und einen komplexen Koeffizienten W^k, die zur Berechnung von zwei neuen Datenpunkten A' und B' Verwendung finden. Der Koeffizient ist ein komplexer Exponentialwert mit der Form

$$e^{-j\Theta} = \cos\Theta - j \sin\Theta.$$

Jede Butterfly-Operation erfordert eine komplexe Multiplikation, eine komplexe Addition und eine komplexe Subtraktion oder vier reale Multiplikationen, drei reale Additionen und drei reale Subtraktionen, wie das in *Bild 1b* dargestellt ist.

Eine FFT wird ausgeführt, indem man die Butterfly-Operationen in einer bestimmten Ordnung miteinander verkettet. Die Butterflies sind in Kolumnen oder Stufen angeordnet; eine N-Punkt-Wurzel-2-FFT besteht aus $\log_2 N$-Stufen, wobei jede N/2 Butterflies umfaßt, so daß sich insgesamt (N/2) $\log_2 N$-Butterflies ergeben. *Bild 2* zeigt die Struktur einer Wurzel-2-DIT-FFT mit 16 Punkten, die vier Stufen mit acht Butterflies, also insgesamt 32 Butterflies umfaßt. Jeder Kreis im Diagramm stellt eine einzelne Wurzel-2-Butterfly-Operation dar. Bei der Bruchzahl, die jeder Butterfly zugeordnet ist, handelt es sich um eine Kurzbezeichnung für den Wert des Koeffizienten, der zur Ausführung dieser Butterfly-Operation

Bild 1. Eine Wurzel-2-DIT-Butterfly-Operation besteht aus drei komplexen Operationen: einer Addition, einer Subtraktion und einer Multiplikation. Die komplexen Eingangsdaten der Punkte A und B erzeugen komplexe Ausgangspunkte A' und B' (a). Diese komplexen Operationen entsprechen vier Multiplikationen, drei Additionen und drei Subtraktionen mit reellen Werten (b)

benötigt wird: Wenn der Wert des Bruches „k" annimmt, beträgt der entsprechende Koeffizientenwert $e^{-j\pi k}$. Die Speicherplätze, in denen die Daten abgelegt sind, werden blockweise zusammengefaßt; im Falle der 16-Punkt-FFT müssen 16 aufeinanderfolgende Speicherplätze für die 16 komplexen Datenpunkte zugewiesen werden. Zum Ausführen jeder FFT-Butterfly-Operation werden die Eingabepunkte vom Datenspeicher geholt und die erforderlichen Operationen mit den entsprechenden Koeffizienten ausgeführt. Anschließend gelangen die Resultate zurück in den Datenspeicher. Die Pfeile in Bild 2 zeigen die Reihenfolge, in der die Butterflies typischerweise abgearbeitet werden. Bei dem gezeigten Algorithmus handelt es sich um eine In-Place-Operation, was bedeutet, daß die von jeder Butterfly-Operation erzeugten Punkte am gleichen Speicherplatz wie die Eingabe-Datenpunkte abgelegt werden.

Der Zugriff auf die Daten erfolgt nicht in der Reihenfolge der Speicherplätze. So muß für das Beispiel der FFT in Bild 2 auf die Daten in der Reihenfolge 0, 8, 1, 9, ..., 7, 15 für die erste Stufe zugegriffen werden. Bei der zweiten und den folgenden Stufen ist die Adressierung noch aufwendiger. So wird die zweite Stufe z. B. durch Zugriff auf die Datenadressen 0, 4, 1, 5, 2, 6, 3, 7 für die erste Gruppe von vier Butterflies ausgeführt; der Zugriff erfolgt dann für die zweite Gruppe der Butterflies auf die Adressen 8, 12, 19, 13, 10, 14, 11, 15. Auch die Butterflies der dritten und vierten Stufen werden in Gruppen adressiert. Für die FFT, die in Bild 2 dargestellt ist, besitzt die Stufe m 2^{m-1} Gruppen von $N/2^m$ Butterflies mit einem Gruppenumfang von $N/2^{m-1}$.

Die Adressierung der Daten stellt allerdings nur eine Hälfte des Problems dar: Auf Koeffizienten muß ebenfalls zugegriffen werden. Für die erste Stufe der FFT in Bild 2 benötigt man lediglich sin(0) und cos(0). Stufe 2 erfordert allerdings die Winkel 0 und $\pi/2$; Stufe 3 benutzt die Winkel 0, $\pi/2$, $\pi/4$, $3\pi/4$; Stufe 4 erfordert die Winkel 0, $\pi/2$, $\pi/4$, $3\pi/4$, $\pi/8$, $5\pi/8$, $3\pi/8$ und $7\pi/8$. Im allgemeinen ist die Koeffizienten-Adreßsequenz für die Stufen m dieses FFT-Typs 0, BR [1] π/N, BR [2] π/N...BR $[2^{m-1} - 1]$ π/N, wobei BR [ ] eine Funktion ist, die die Reihenfolge der Bits in einem Binärwort umkehrt. Für jede Gruppe Butterflies muß auf einen neuen Koeffizienten zugegriffen werden.

Häufig verwendete Lösungen

Unterschiedliche FFT-Typen erfordern verschiedene Arten der Adressierung, allerdings zeigt das Beispiel einen repräsentativen Typ des sequentiellen Adreßzugriffs. Entwickler von FFT-Systemen haben unterschiedliche Techniken herangezogen, um diese FFT-Daten- und Koeffizienten-Adressen zu erzeugen, wobei ihnen allerdings unterschiedlicher Erfolg beschieden war.

Eine der üblichen Lösungen des Adressierungsproblems ist, Daten- und Koeffizienten-Adressen in PROMs abzulegen und diese sequentiell mit einem Zähler zu adressieren. Diese Lösung hat allerdings mehrere schwere Nachteile: Erstens wird die Anzahl der Adressen unverhältnismäßig groß, wenn der Umfang der FFT zunimmt. Eine Wurzel-2-FFT mit 4096 Punkten und komplexen Eingangswerten muß beispielsweise 24 576 Butterflies adressieren, die jeweils zwei Datenadressen und eine Koeffizientenadresse erfordern, so daß sich insgesamt 73 728 Adressen ergeben. Während sich die Anzahl der Adressen durch einen konstanten Geometrie-FFT-Algorithmus, der gleiche Datenadressen für jede Stufe verwendet, reduzieren lassen, hat dieser

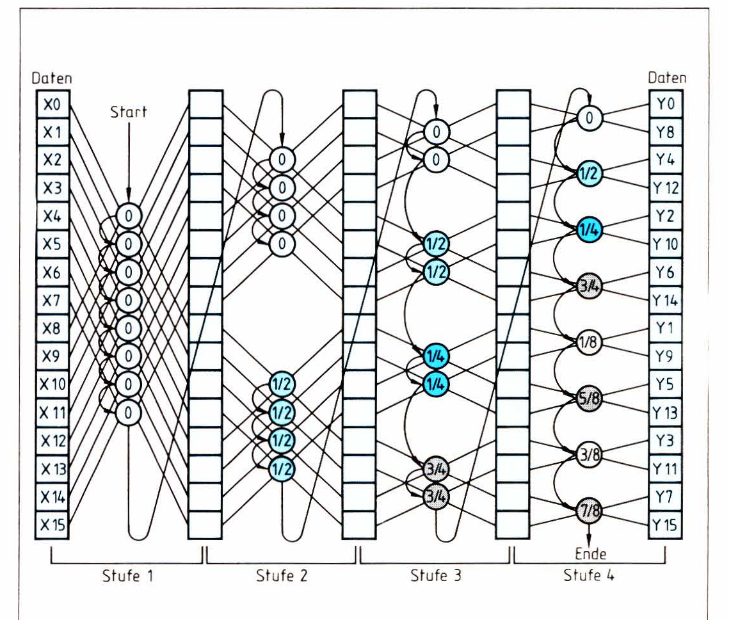


Bild 2. Diese 16-Punkte-Wurzel-2-DIT-FFT enthält 32 Butterflies. Die Pfeile zeigen die Reihenfolge, in der die Butterflies typischerweise abgearbeitet werden

Algorithmus den Nachteil, daß er sich nicht für das In-Place-Verfahren eignet, so daß im Endeffekt die doppelte Datenspeicherkapazität erforderlich ist. Der zweite Nachteil der PROM-Lösung ist, daß wenn ein einziges System verschiedene Größen oder Typen der FFT ausführen muß, jeweils eine andere Adreßtabelle für jede FFT erforderlich ist. Dies ist oft der Fall bei programmierbaren digitalen Signalprozessoren.

Eine weitere häufig verwendete Methode ist die Lösung des Adressierungsproblems mit umfangreichen Systemen aus MSI- und SSI-Schaltungen. Diese Lösung läßt sich relativ einfach implementieren, führt aber in der Regel zu einer Schaltung, die viel Platinenfläche einnimmt, Schwierigkeiten bei der Steuerung verursacht und nicht einfach fehlerfrei zu machen ist. Eine Schaltung, die die Adressierfunktion für eine FFT mit 4096 Punkten vornimmt, kann aus 10...20 MSI- und SSI-Chips bestehen.

Die dritte Möglichkeit ist die Berechnung der erforderlichen Adressen mit Hilfe von Software, eine Methode, die allerdings in vielen Fällen zu langsam für Echtzeit-Anwendungen ist.

Adreß-Sequenzer Am29540

Der FFT-Adreß-Sequenzer Am29540 übernimmt die Adressierung für eine Anzahl von FFT-Algorithmen. Das Bauelement läßt sich in drei Abschnitte unterteilen: Butterfly-Zähler, Daten-Adreßgenerator und Koeffizienten-Adreßgenerator (*Bild 3*).

Der Butterfly-Zähler besteht in Wirklichkeit aus zwei Zählern: einer für Spalten und einer für Zeilen. Der Spaltenzähler zeigt auf die derzeit bearbeitete FFT-Stufe oder auch Spalte, während der Zeilenzähler auf die Butterfly zeigt, die derzeit innerhalb der Stufe ausgeführt wird. Beide Zähler sind programmierbar und lassen sich so initialisieren, daß sie Transformationen verschiedener Länge ausführen, indem sie den entsprechenden 4-Bit-Transformationslängen-Code in einem Zwischenspeicher auf dem Chip ablegen, bevor die Transformation beginnt. Zur Eingabe wird dieser Code an die Eingangsleitungen TL0...3 gelegt und mit den Signalen TSEL und TSRB in den Zwischenspeicher übernommen.

Der Butterfly-Zähler führt einen von vier Befehlen aus: Reset/Load, Zählen und Halten. Diese Befehle werden mit Hilfe der Steuerleitungen I0...1 ausgewählt und mit der ansteigenden Flanke des Takteingangs CP ausgeführt. Eine FFT wird durch Initialisierung des Butterfly-Zählers mit einem Reset oder einer Reset/Load-Instruktion begonnen. Die Zähl- und Halte-Instruktionen dienen dazu, den Zähler auf die nächste Butterfly-Operation zu schalten oder diesen auf der derzeitigen Operation zu halten.

Der Butterfly-Zähler erzeugt vier Flags, mit deren Hilfe die FFT-Sequenz ausgeführt wird. „IT COMP“ (Iteration Complete) zeigt die letzte Butterfly in einer Stufe oder Spalte an; die letzte Butterfly in einer FFT wird durch „FFT COMP“ (FFT Complete) gemeldet. Das Flag EVEN/$\overline{\text{ODD}}$ verändert seinen Zustand nach jeder Stufe und läßt sich dazu benutzen, die Speicheroperationen für Non-In-Place-Transformationen zu steuern. Das vierte Flag, KNZ/$\overline{\text{KZ}}$, kommt insbesondere dann zur Anwendung, wenn Transformationen mit realen Eingangsvariablen ausgeführt werden. EVEN/$\overline{\text{ODD}}$ und KNZ/$\overline{\text{KZ}}$ werden im Multiplex auf einen einzigen Anschluß umgeschaltet. Wenn der Am29540 eine Datenadresse für einen Realwert produziert, erscheint KNZ/$\overline{\text{KZ}}$ auf dem Stift, für jede andere Adresse EVEN/$\overline{\text{ODD}}$. Die Daten- und Koeffizienten-Adressengeneratoren erzeugen die Adressen kombinatorisch, wobei sie den Stand des Butterfly-Zählers als Eingangssignal nehmen. Mit Hilfe des Datenadreßgenerators werden die Adressen für die jeweiligen Ein- und Ausgangs-Datenpunkte einer Butterfly erzeugt, während der Koeffizienten-Adreßgenerator Adressen für Koeffizienten und Wichtungsfunktionen produziert. Drei Steuersignale – $\overline{\text{PSD}}$, DIT/$\overline{\text{DIF}}$ und RADIX4/$\overline{2}$ – konfigurieren die Adreßgeneratoren für die verschiedenen FFT-Typen. Diese Signale sind in einem Latch des Speichers abgelegt, die mit Hilfe der Freigabe-Leitungen $\overline{\text{SEL}}$ und $\overline{\text{STRB}}$ gesteuert werden. Ein Benutzer wählt die gewünschten Eingangsdaten, Ausgangsdaten oder Koeffizientenadressen mit Hilfe der Steuerleitungen AS0...3 aus. Die AS-Werte 0...7 und 12...15 wählen Datenadressen, die Werte 8...11 wählen Koeffizientenadressen. Von AS0...3 wird eine Adresse gewählt, die auf den Adreßleitungen A0...15 erscheint. Diese Adreßleitungen lassen sich mit Hilfe des Signals $\overline{\text{OE}}$ auf hohe Impedanz schalten, so daß sich mehrere Bausteine zur Adreßerzeugung auf dem gleichen Bus zusammenschließen lassen.

Bild 3. Der Baustein Am29540 ist die erste Ein-Chip-Lösung für das Problem der FFT-Adressierung. Ein interner Butterfly-Zähler steuert die Daten- und Koeffizienten-Adreßgeneratoren, die wiederum die Adressen für die unterschiedlichsten FFT-Algorithmen erzeugen

Adressierung der Daten

Wenn Daten adressiert werden, kann der Am29540 nicht weniger als 2^{16} Adressen erzeugen; die tatsächliche Anzahl der Datenadressen für eine bestimmte FFT hängt von der Größe und dem Typ der Transformation ab. Eine Wurzel-2-In-Place-FFT mit N Punkten und komplexen Eingangswerten muß beispielsweise

N komplexe Datenpunkte während jeder Stufe der Transformation adressieren. Wenn N die Zahl 16 annimmt, wie das bei der FFT in Bild 2 der Fall ist, müssen 16 Speicherplätze adressiert werden, so daß ein großer Teil des verfügbaren Adreßbereiches ungenutzt bleibt.

Der Am29540 bietet zwei Datenadressierungs-Optionen für denjenigen Benutzer, der weniger als 64-K-Worte Adreßbereich benötigt. Die erste Option setzt die nichtbenutzten oberen Adreßbits für die Daten auf 0; dies erreicht man durch Initialisierung des Butterfly-Zählers mit der Reset-Instruktion. Für die 16-Punkte-Transformation nach Bild 2 kann man daher die oberen 12 Adreßleitungen für jede Datenadresse auf 0 setzen. Die andere Option ist die Programmierung der oberen Adreßleitungen auf einen Wert, den der Benutzer festlegt, so daß ein Mittel zur Verfügung steht, verschiedene Datenblocks in einem großen Speicher zu adressieren. Die oberen Datenbits werden programmiert, indem man OE auf High-Pegel legt und das gewünschte Bitmuster auf die Adreßleitungen A0...15 schaltet. Danach muß man die Reset/Load-Instruktion des Butterfly-Zählers ausführen. Wenn z. B. das Bitmuster $ABCO_{(16)}$ zur Initialisierung einer komplexen FFT mit 16 Punkten benutzt wird, adressiert der Am29540 einen Block von 16 Datenplätzen, der bei der Adresse $ABCO_{16}$ beginnt.

Non-In-Place-Transformationen führen zu zusätzlichen Problemen bei der Adreßerzeugung. Im Unterschied zu In-Place-Transformationen können Non-In-Place-Algorithmen die Ausgangsdaten einer Butterfly-Operation nicht auf dem gleichen Speicherplatz, der vorher von den Eingangsdaten verwendet wurde, abgelegt werden. Mit Hilfe des Am29540 läßt sich dieses Problem überwinden, weil dieser Baustein sowohl die Ein- als auch die Ausgangs-Datenadressen für diese Transformationen erzeugt. Typischerweise werden Non-In-Place-Transformationen mit Hilfe von zwei Datenspeichern ausgeführt, dabei ist einer die Quelle für die Eingangsdaten, der andere der Bestimmungsort für die Ausgangsdaten. Wenn eine Stufe ausgeführt ist, vertauscht man die Rolle dieser Speicher, so daß die Ausgangsdaten einer Stufe die Eingangsdaten der nächsten Stufe sind. Das Signal EVEN/$\overline{\text{ODD}}$ ist insbesondere für diesen Fall sehr hilfreich, weil es nach jeder Stufe seinen Zustand verändert. Es läßt sich daher zur Steuerung der Richtung des Datenflusses zwischen den beiden RAMs verwenden.

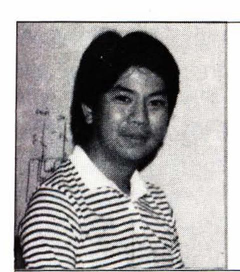
Tabelle der Transformationstypen, die vom AM29540 unterstützt werden

Eingangs-Datentyp	Wurzel	Dezimierungs-Typ	In-Place/ Non-In-Place	Eingangs-Daten	Ausgangs-Daten	Richtung der Transformation
	2	DIF	In-Place	Normal	Rückwärts	Vorwärts/Invers
	2	DIF	In-Place	Rückwärts	Normal	Vorwärts/Invers
	2	DIF	Non-In-Place	Normal	Normal	Vorwärts/Invers
	2	DIT	In-Place	Normal	Rückwärts	Vorwärts/Invers
	2	DIT	In-Place	Rückwärts	Normal	Vorwärts/Invers
Komplex	2	DIT	Non-In-Place	Normal	Normal	Vorwärts/Invers
	4	DIF	In-Place	Normal	Rückwärts	Vorwärts/Invers
	4	DIF	In-Place	Rückwärts	Normal	Vorwärts/Invers
	4	DIF	Non-In-Place	Normal	Normal	Vorwärts/Invers
	4	DIT	In-Place	Normal	Rückwärts	Vorwärts/Invers
	4	DIT	In-Place	Rückwärts	Normal	Vorwärts/Invers
	4	DIT	Non-In-Place	Normal	Normal	Vorwärts/Invers
Reale	2	DIT	In-Place	Normal		Vorwärts
Werte	2		In-Place		Normal	Invers

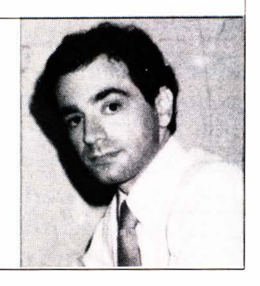
Adressierung der Koeffizienten

Um auf Koeffizienten zuzugreifen, erzeugt der Am29540 eine 16-Bit-Adresse, die einem von 2^{16} Winkeln zwischen 0 und 2 π entspricht. Für die Koeffizientenadresse A ist der adressierte Winkel $2\pi A/2^{16}$; der Winkel $\pi/2$ hat beispielsweise die Adresse 4000_{16}. Die Koeffizientenadresse wird an den Look-Up-Speicher angelegt, üblicherweise ein PROM, das Sinus- und Cosinus-Werte für die gewählten Winkel enthält.

Eine bestimmte FFT benötigt in der Regel nur eine gewisse Auswahl aus den 64 K verfügbaren Winkelwerten. So benötigt eine Wurzel-2-FFT für N Punkte mit komplexen Eingangswerten Zugriff auf N/2 Winkel mit gleichen Abständen im Bereich von 0...π; das in Bild 2 gezeigte Beispiel mit einer 16-Punkt-FFT benötigt lediglich acht verschiedene Winkel. Der Baustein Am29540 wählt automatisch die erforderlichen Winkel in der richtigen Reihenfolge aus und überspringt nichtverwendete Werte.



David Quong ist Produktplanungs-Ingenieur und zuständig für den Entwurf von Produkten aus dem Bereich digitaler Signalprozessoren und Array-Prozessoren bei Advanced Micro Devices. Er erwarb seinen BSEE an der California State University in Sacramento.



Robert M. Perlman ist zuständig für Produktplanung in der Abteilung für Array-Prozessoren bei Advanced Micro Devices. Seinen MSEE erhielt er an der John's Hopkins University im Jahre 1981. Bevor er zu AMD kam war er unter anderem bei Westinghouse beschäftigt.

Das Koeffizienten-Adressierverfahren, das zur Anwendung kommt, bietet einen wichtigen Vorteil für Systeme, auf denen unterschiedliche Größen von FFT-Operationen implementiert sind. Weil der Am29540 automatisch lediglich auf diejenigen Sinus- und Cosinus-Werte zugreift, die erforderlich sind, kann eine einzige Sinus/Cosinus-Tabelle zur Ausführung verschiedener unterschiedlich großer FFT-Operationen Verwendung finden. Wenn z. B. der Benutzer eine Look-Up-Tabelle mit 2048 Sinus- und Cosinus-Werten zwischen 0 und π vorsieht, kann die Tabelle zur Ausführung aller komplexen Wurzel-2-FFTs mit 4096 oder weniger Punkten Verwendung finden.

Allen FFTs gewachsen

Weil jeder, der eine FFT implementieren muß, einem individuellen Spektrum an Entwurfskriterien gegenübersteht, ist es nicht verwunderlich, daß heute eine ganze Anzahl von FFT-Algorithmen angewendet wird. Die Programmierbarkeit des Am29540 ermöglicht es einem Benutzer, Adressen für eine Vielzahl von FFT-Typen zu erzeugen.

Die meisten FFT-Algorithmen, die derzeit verwendet werden, sind für komplexe Eingangsdaten ausgelegt. Der Typ Am29540 unterstützt 12 verschiedene Typen dieser Transformationen (*Tabelle*). Unter anderem stehen dem Entwickler folgende zur Verfügung:

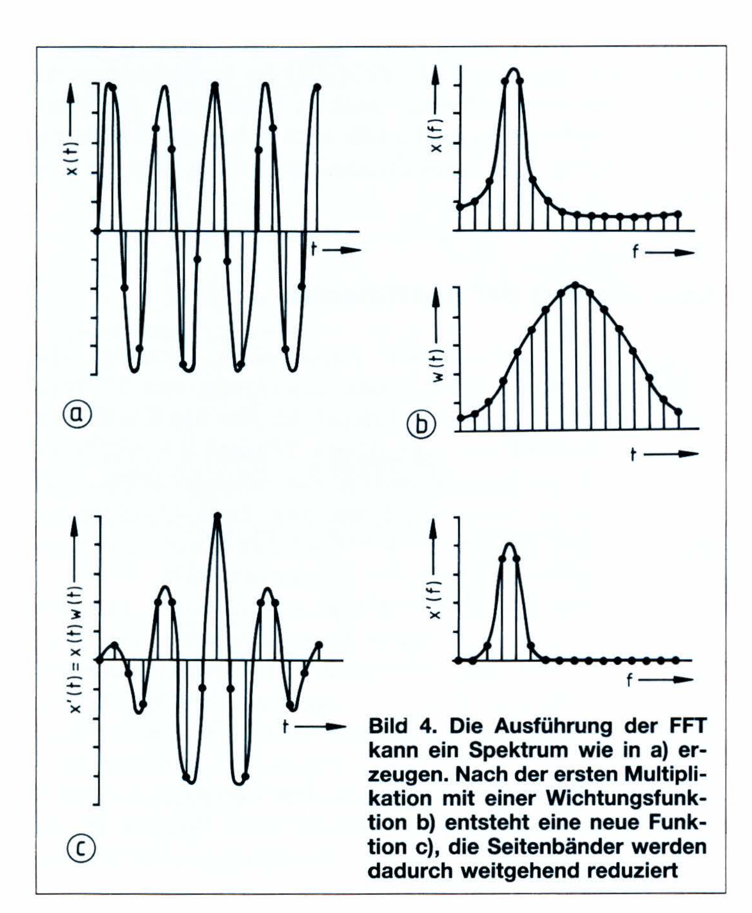


Bild 4. Die Ausführung der FFT kann ein Spektrum wie in a) erzeugen. Nach der ersten Multiplikation mit einer Wichtungsfunktion b) entsteht eine neue Funktion c), die Seitenbänder werden dadurch weitgehend reduziert

Wurzel-2- oder Wurzel-4-Transformationen – die Butterfly-Struktur einer Wurzel-4-Transformation ist wesentlich komplizierter als diejenige für Wurzel 2, allerdings bietet sie eine etwas größere Berechnungseffizienz. Jede Wurzel-4-Butterfly erzeugt vier Ausgangsdatenpunkte aus vier Eingangsdatenpunkten und drei Koeffizienten, erforderlich sind 12 Multiplikationen und 22 Additionen für Realzahlen. Wurzel-4-Transformationen werden mit Hilfe des Signals RADIX4/$\overline{2}$ gewählt.

Decimation in Time (DIT) oder Decimation in Frequency (DIF) – diese beiden Begriffe beziehen sich auf zwei FFT-Klassen; die Namen geben eine Aussage, in welcher Weise die Werte abgeleitet werden. DIT- und DIF-Butterfly-Strukturen unterscheiden sich etwas, erfordern allerdings die gleiche Anzahl arithmetischer Operationen. Man wählt die gewünschte Transformationsart mit Hilfe des Signals DIT/$\overline{\text{DIF}}$.

In-Place- oder Non-In-Place-Transformationen – wie bereits erwähnt, benötigt Non-In-Place-Transformation einen doppelt so großen Datenspeicher im Gegensatz zum In-Place-Verfahren. Man könnte also annehmen, daß daher In-Place-Transformationen bevorzugt werden. Allerdings hat das In-Place-Verfahren einen Nachteil: Entweder die Eingangs- oder Ausgangs-Daten der Transformation müssen in umgekehrter Zählweise der Adressen im Speicher abgelegt sein, so daß anschließend eine Neusortierungs-Operation auszuführen ist. Zwischen In-Place- und Non-In-Place-Algorithmen schaltet man um, indem entsprechende Werte von AS0...3 die gewünschten Adressen wählen. Sollte der Benutzer eine In-Place-Transformation wählen, muß er mit dem Signal PSD die Ein- oder Ausgangs-Werte in umgekehrter Adreßreihenfolge wählen.

Der Baustein Am29540 läßt sich auch zur Ausführung inverser Transformationen verwenden, ein nützliches Merkmal in Anwendungen, bei denen man den Übergang von der Frequenz- zur Zeitdomäne benötigt. Die Berechnung inverser Transformationen erfolgt gradlinig – die Adreßsequenzen, die benötigt werden, sind die gleichen wie bei Vorwärtstransformation. Bei Wurzel-2-Transformationen ist der einzige Unterschied zwischen inverser und Vorwärtstransformation die komplexe Exponentialfunktion: $e^{-j\Theta}$ muß ersetzt werden durch $e^{j\Theta}$. Der Austausch des Vorzeichens im Argument der Exponentialfunktion entspricht dem Ersatz des Koeffizienten $\sin\Theta$ durch $-\sin\Theta$, eine Operation, die einfach durch Modifizierung der Additionen und Subtraktionen, die innerhalb der Butterfly auftreten, auszuführen ist. Wurzel-2-Transformationen erfordern in ähnlicher Weise kleinere Veränderungen von Vorzeichen in der Butterfly-Berechnung, um inverse Transformationen auszuführen.

Einige Anwendungen erfordern FFT-Transformationen mit Realzahlen. Der Am29540 erzeugt Datenadressen für Vorwärts- und Invers-Transformationen mit realen Eingangswerten (RVI) vom Typ, der erstmals von Bergland beschrieben worden ist [1]. Diese Transformationen sind in der Tabelle aufgelistet.

Wichtung

FFT-Filtercharakteristika lassen sich häufig merklich verbessern, indem die Eingangsdaten vorher mit Faktoren multipliziert werden (*Bild 4*). Diese Operation wird auch Wichtung, Windowing oder Shading genannt. Verschiedene übliche Wichtungsfunktionen sind bereits dokumentiert [2].

Der Baustein Am29540 unterstützt zwei Wichtungsverfahren für Wurzel-2-Transformation. Das erste und einfachste ist die Durchführung eines Durchlaufes, bevor die FFT beginnt. Dazu werden Adressen erzeugt, indem der Am29540 auf die erste Stufe der Wurzel-2-DIF-Transformation programmiert wird. Dabei entstehen Datenadressen, die zum Zugriff auf Eingangsdaten benötigt werden sowie die Koeffizienten zum Zugriff auf die Wichtungswerte, die in einer Look-Up-Tabelle gespeichert sind. Nach Beendigung des Vorlaufs wird der Baustein neu programmiert, um die gewünschten Werte für eine Wurzel-2-FFT zu adressieren.

Die zweite Möglichkeit nutzt die Vorteile der Struktur einer DIT-FFT. In der ersten Stufe einer DIT-FFT sind lediglich die Werte Sinus 0 und Cosinus 0 erforderlich. Daraus ergibt sich, daß alle Multiplikationen der ersten Stufe mit Faktoren von +1, −1 oder 0 erfolgen, so daß ein Multiplizierer für diesen Vorgang eigentlich gar nicht erforderlich ist. Die Wichtung kann daher, wenn sie in dieser Stufe untergebracht werden soll, den ungenutzten Multiplizierer verwenden. Indem man den Am29540 so konfiguriert, daß er eine Wurzel-2-DIF-FFT für die Stufe 1 ausführt und dann den FFT-Typ von DIF auf DIT für alle übrigen Stufen ändert, lassen sich die erforderlichen Daten, Wichtungen und Koeffizientenadressen erzeugen.

Praktisch alle brauchbaren Wichtungsfunktionen sind symmetrisch um einen zentralen Punkt. Wenn Y (n) eine symmetrische N-Punkte-Wichtungsfunktion ist, muß der Punkt Y (x) mit dem Punkt Y (N-x) übereinstimmen. Diese Symmetrie bedeutet, daß ein Benutzer nicht alle N Punkte der Wichtungsfunktion zu speichern hat: (N/2) + 1 Punkte sind ausreichend. Der Typ Am29540 unterstützt die Adressierung solcher halber Tabellen, indem gleichzeitig die Adressen s und m-x erzeugt werden, wodurch sich die erforderliche Speicherkapazität für die Wichtungsfunktionen verringert. Aus diesem Grund ist ein separater Speicher für die Wichtungsfunktion (wie bei *van Hann*) nicht erforderlich.

Typische FFT

Der Betrieb des Am29540 läßt sich am besten verstehen, wenn man dessen Verwendung in einer typischen FFT betrachtet. Wenn man beispielsweise eine 16-Punkt-DIT-FFT (Bild 2) implementieren möchte, entspricht dieser Typ der vierten Zeile von Tabelle 2. Um den Am29540 zu initialisieren, muß man die entsprechenden Transformationslängen-Codes und Steuerbits in die Latches des Chips laden. Bei diesem Beispiel hat der Transformationslängen-Code einen Wert von 0011_2; Steuerbits haben die Werte $\overline{PSD}$ = 1, RADIX4/$\overline{2}$ = 0 und DIT/$\overline{DIF}$ = 1. Nach der Programmierung des Bausteins mit diesen Daten wird der Butterfly-Zähler mit der Reset- oder Reset/Load-Instruktion initialisiert.

Nachdem er initialisiert ist, erzeugt der Baustein Daten- und Koeffizienten-Adressen für die erste Butterfly-Operation des FFT. Anschlüsse AS0...3 wählen die verschiedenen Adressen aus. Für diesen Algorithmus werden die Eingangs- und Ausgangs-Datenadressen für AS = 0, 1 produziert, die Koeffizientenadressen für AS = 8. Nachdem alle Adressen gelesen wurden, geht der Baustein zur nächsten Butterfly über, indem eine Zählinstruktion ausgeführt wird. Die Reihenfolge, in der die Butterflies abgearbeitet werden, zeigt der Pfeil in Bild 2. Der zeitliche Ablauf, die Initialisierung und die sequentielle Erzeugung der Adressen sind in *Bild 5* dargestellt. Es ist zu beachten, daß die in Bild 2 dargestellte FFT-Einrichtung die Ausgangsdaten in umgekehrter Reihenfolge der Adressen erzeugt.

Literatur

[1] *Bergland, G. D.:* A Fast Fourier Transform Algorithm for Real-Valued Series. Communications of the ACM, Oktober 1968, Seite 703...710.

[2] *Harris, F. J.:* On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform. Proceedings of the IEEE, Januar 1978, Seite 51...83.

[3] *Renz, U., New, B. J.:* Bausteinfamilie vereinfacht FFT. EELEKTRONIK 1982, H. 21, S. 127...132.

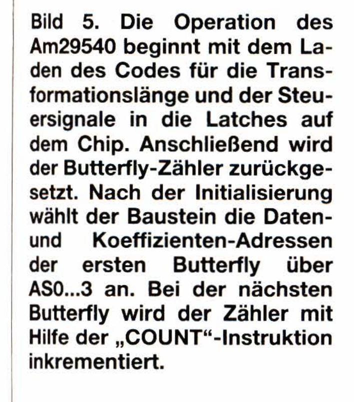
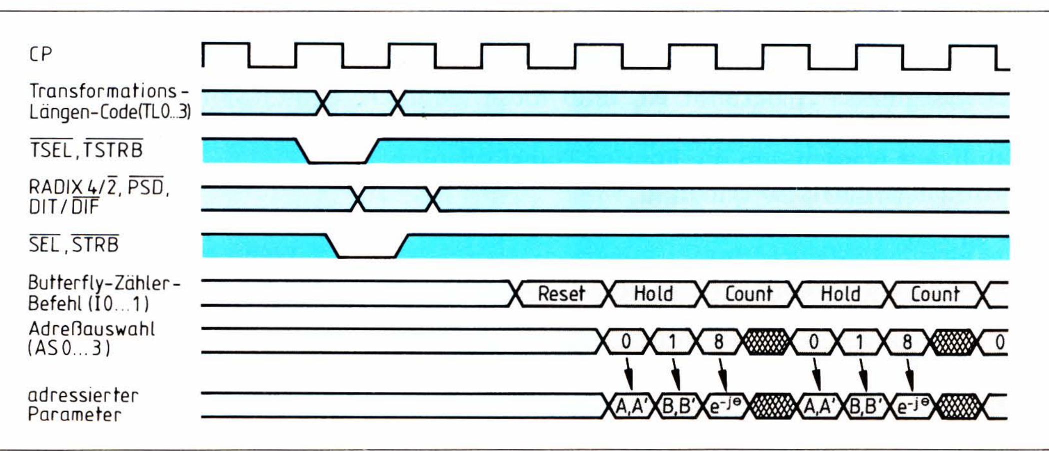


Bild 5. Die Operation des Am29540 beginnt mit dem Laden des Codes für die Transformationslänge und der Steuersignale in die Latches auf dem Chip. Anschließend wird der Butterfly-Zähler zurückgesetzt. Nach der Initialisierung wählt der Baustein die Daten- und Koeffizienten-Adressen der ersten Butterfly über AS0...3 an. Bei der nächsten Butterfly wird der Zähler mit Hilfe der „COUNT"-Instruktion inkrementiert.



Udo Renz, Bernard J. New

Bausteinfamilie vereinfacht FFT

Beim Entwurf von Signalverarbeitungssystemen muß man komplexe Vorüberlegungen anstellen. Die wichtigste Entscheidung in diesem Zusammenhang ist, ob analoge oder digitale Techniken Verwendung finden sollen. Dann muß man festlegen, ob die Verarbeitung im Zeit- oder im Frequenzbereich durchzuführen ist. In der Vergangenheit sorgte die verfügbare Hardware dafür, daß die Entscheidung meistens zugunsten des Zeitbereichs ausfiel. Inzwischen ist aber eine neue IC-Familie auf den Markt gekommen, die es dem Entwickler erleichtert, im Frequenzbereich eine optimale Systemlösung zu finden.

1 Faltung und Frequenzbereich

Am Beispiel eines der verbreitetsten Signalverarbeitungs-Algorithmen, der Faltung, wird deutlich, warum man bisher im Zeitbereich arbeitete, obwohl es im Frequenzbereich wesentlich effizienter ist. Dieses Verfahren wird dazu benutzt, die Übertragungsfunktion Y eines beliebigen linearen, zeitlich invarianten Systems in bezug auf das Eingangssignal X zu bestimmen. In der zeitdiskreten Form (also digital) wird diese Funktion durch folgende Summierung wiedergegeben:

$$Y_k = \sum_{i=o}^{M-I} h_i \, X_{k-i}$$

Die Ausdrücke h_i, i=o bis M−I geben das Impulsverhalten des Systems an. Um die Ausgangswerte zu erhalten, müssen jeweils M Multiplikationen ausgeführt und die Produkte addiert werden. M muß eine endliche Zahl sein, daher werden die auf diesem Verfahren beruhenden Anordnungen auch „Finite-Impulse-Response-Filter“ (FIR) genannt. Um einen Filter hoher Qualität zu realisieren, muß die Zahl M sehr groß sein, was zu einer umfangreichen Rechenbelastung des Systems führt.

Im Frequenzbereich wird die Faltung durch Multiplikation ersetzt [2]. Wenn die Fourier-Transformation des Eingangssegments X bekannt ist, muß diese lediglich punktweise mit der Übertragungsfunktion des Systems multipliziert werden, um die Fourier-Transformation Y des Ausgangssignals zu erhalten.

$$y_k = H_k \cdot X_k$$

Das Frequenzverhalten bzw. die Übertragungsfunktion H_k ist die Fourier-Transformation des Impulsverhaltens. Es ist bei diesem Verfahren pro Punkt lediglich eine komplexe Multiplikation (vier im reellen Zahlenbereich) erforderlich, wodurch das Verfahren eine hohe Effizienz aufweist. Allerdings müssen im Normalfall Ein- und Ausgangswerte im Zeitbereich vorliegen. Daher ist es notwendig, die Fourier-Transformation auszuführen. Einen kontinuierlichen Datenstrom erreicht man, indem die Menge der Daten in Blocks aufgeteilt und die diskrete Fourier-Transformation (DFT) ausgeführt wird. Die Transformation der Ein- und Ausgangswerte nimmt den größten Aufwand in Anspruch, weil das Impulsverhalten lediglich einmal zu transformieren ist. Wenn die Daten in N Blocks (N Stützstellen) aufgeteilt werden, sind N^2 komplexe Multiplikationen pro Transformation erforderlich. Das ergibt eine Gesamtzahl von 8 N + 4 reellen Multiplikationen. Weil N größer als M gewählt werden muß, um die Transformation des Impulsverhaltens zu ermöglichen, wird klar, daß dieses Verfahren weniger effizient als die Verarbeitung im Zeitbereich ist.

Die Situation verbessert sich bei Anwendung der schnellen Fourier-Transformation (FFT). Diese reduziert die Anzahl der erforderlichen Multiplikationen pro Transformation von N^2 auf $\frac{1}{2}$ N · $\log_2$ N.

Die Zahl der reellen Multiplikationen beträgt dann pro Ausgangs-Stützstelle 4 · $\log_2$ N+4. Die Zahl kann kleiner als M gemacht werden. Diese einfache Analyse berücksichtigt allerdings noch nicht, daß die Faltung zyklisch durchgeführt wird. Aus diesem Grunde beträgt die Anzahl der Multiplikationen pro Stützstelle (*Bild 1*):

$$\frac{4\ N \log_2 N+4N}{N-M+1}$$

Wenn man z. B. N = 1024 wählt, werden 101 Stützstellen mit 49 reellen Multiplikationen berechnet. Das sind weniger als die Hälfte im Vergleich zur Methode im Zeitbereich, bei der 101 erforderlich sind. Wenn zwei Signale durch identische Filter geleitet werden, kann das ohne zusätzliche Multiplikationen erfolgen. Der Frequenzbereich ist daher trotzdem noch wirtschaftlich.

2 Problem FFT

Trotz verbesserter Effizienz wird die Faltung am häufigsten im Zeitbereich durchgeführt, weil Prozessoren, die für FFT Verwendung finden, sehr komplex sind. Diese Komplexität besteht in zweifacher Hinsicht: Erstens muß der Prozessor komplexe Zahlen verarbeiten. Die meisten Prozessoren sind in der Lage, dies zu tun, allerdings müssen Real- und Imaginärteil im Multiplexbetrieb verarbeitet werden. Für die meisten Fälle ist das nicht optimal, weil die Parallelstruktur der Spezial-Prozessoren nicht ausgenutzt wird. Allerdings sind die derzeit verfügbaren LSI-Schaltungen nicht so strukturiert, daß sie zu einer solchen Prozessor-Architektur passen, und ein Entwickler möchte nicht unbedingt einen eigenen aus MSI- und SSI-Schaltungen aufbauen. Das zweite und wesentlich größere Problem ist die Erzeugung der Adreßsequenzen. Die FFT besteht aus einer wiederholten Implementation einer Kernoperation mit der Bezeichnung „Butterfly“. Diese ist relativ einfach und umfaßt nur eine komplexe Multiplikation, eine komplexe Addition sowie eine komplexe Subtraktion, die allerdings für eine Transformation mit N Stützstellen

$$N/2 \cdot \log_2 N$$

mal ausgeführt werden müssen, d. h. beispielsweise für 1024 Punkte insgesamt 5120mal. Jede Butterfly erfordert zwei Datenadressen sowie die Adresse einer komplexen Konstantentabelle. Normalerweise werden diese Adressen in einem separaten MSI-Prozessor erzeugt, der parallel zum Arithmetikprozessor arbeitet und aus 20...40 ICs besteht.

Durch diese Tatsache wird die Verwendungsmöglichkeit für die FFT stark eingeschränkt. So ist z. B. bei der Spektrumanalyse die DFT wesentlich weniger effizient als die FFT. Trotzdem wird sie in schmalbandigen Systemen oft benutzt, obwohl die Anzahl der Ausgangsleitungen normalerweise für die Verwendung der FFT sprechen würde. Viele andere Systeme arbeiten in unwirtschaftlicher Weise, nur weil es zu schwierig ist, in den Frequenzbereich überzugehen. Eine schaltungstechnisch einfache Lösung würde dem Entwickler ein leistungsfähiges Werkzeug in die Hand geben.

3 Die Bausteinfamilie Am29500

Zur Lösung der hier angesprochenen Probleme bei der Signalverarbeitung wurde die Bausteinfamilie Am29500 konzipiert. Die wichtigsten Mitglieder sind mikroprogrammierte LSI-Funktionen unter Benutzung der 8-Bit-Byte-Slice-Architektur. Wo es sinnvoll erschien, wurden nicht erweiterbare Funktionen hinzugefügt, z. B. MSI-Unterstützungsfunktionen. Alle Bausteine sind intern in ECL-Technik aufgebaut, für die E/A-Anschlüsse werden die Pegel auf Werte umgesetzt, die TTL-kompatibel sind. Diese Schaltungstechnik sorgt zusammen mit dem IMOX-Prozeß (Oxidisolation) für hohe Systemgeschwindigkeit.

Zwei Mitglieder der Bausteinfamilie, der mikroprogrammierbare Signalprozessor Am29501 und die 16-Bit-Parallelmultiplizierer Am29516/29517, ermöglichen die Konstruktion von effizienten, parallelen Arithmetikprozessoren mit Pipeline-Struktur. Sie eignen sich insbesondere für die wiederholten kurzen Operationen der Signalverarbeitung wie z. B. schnelle Multiplikation und Manipulation komplexer Zahlen.

Das zweite Problem, die FFT-Adreßgeneration, wird vom Adreßsequenzer Am29540 gelöst. Dieser 40polige Baustein erzeugt alle Adressen, die für einen weiten Bereich von FFT-Variationen notwendig sind, und ist in seiner Transformationslänge programmierbar. Der Systemdurchsatz wird durch eine überlappende Struktur mit Pipeline-Technik optimiert und durch zwei Multi-Level-Pipeline-Register (Am29520 und Am29521) unterstützt.

Mit der Familie Am29500 kann man einen leistungsfähigen Array-Prozessor, wie er in *Bild 2* dargestellt ist, konstruieren. Dieser führt, als Slave-Prozessor am Hauptbus des Systems

Bild 1. Rechenbelastung für Faltung im Zeit- und Frequenzbereich

angeschlossen, eine große Zahl von Arithmetik-Operationen aus, die überwiegend aus Signalverarbeitungsalgorithmen bestehen. Bei Verwendung dieser Architektur ist es möglich, ein Wurzel-2-Butterfly in vier Instruktionszyklen zu implementieren. Mit den Leistungsmerkmalen, die für die Bausteine angegeben werden, kann eine komplexe FFT mit 1024 Stützstellen in weniger als 3 ms ausgeführt werden.

4 FFT-Adreßsequenzer Am29540

Weil die Arbeitsweise des FFT-Adreßsequenzers Am29540 stark mit den Grundlagen dieser Technik zusammenhängt, kann die Beschreibung nur in der speziellen FFT-Terminologie erfolgen. Für Leser, denen diese Technik noch nicht geläufig ist, folgt eine kurze Zusammenfassung. Nähere Einzelheiten finden sich unter den Literaturstellen [2] und [3].

Eine FFT besteht aus vielen „Butterfly"-Operationen. In der Regel haben diese zwei komplexe Eingangswerte, die mit einem komplexen Koeffizienten kombiniert werden, wobei sich zwei komplexe Ausgangswerte ergeben:

$$A' = A + BW_N^k$$
$$B' = A - BW_N^k \qquad W_N = E^{\frac{-2\frac{1}{2}J}{N}}$$

Dabei ist N die Anzahl der transformierten Punkte, die in diesem Fall eine Potenz von 2 sein muß.

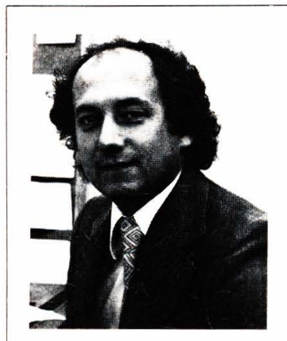
Die Operation wird so oft durchgeführt, bis alle N Eingangspunkte verarbeitet sind und ein neuer Satz von N Punkten, genannt „Durchgang" oder „Kolumne" von Butterflies, erzeugt ist. Die gesamte Operation wird mit den neuen Daten ausgeführt, wobei die Paarung anders gewählt ist. Bei 2^n Punkten müssen n Durchgänge von N/2 Butterflies gemacht werden, um die Daten fortschreitend zu transformieren. Insgesamt werden

$$\tfrac{1}{2}\, N \cdot \log_2 N$$

Butterflies ausgeführt.

Eine wichtige FFT-Klasse ist in *Bild 3* gezeigt. Es handelt sich um eine 8-Punkt-Transformation, für die drei Kolumnen vorgesehen sind, die jeweils vier Butterfly-Operationen ausführen. Zunächst enthalten die acht Speicherplätze Eingangsdaten. Dann durchlaufen Paare der Datenpunkte die Butterfly-Operation und die Ausgangswerte in die Speicherplätze zurückgeschrieben, aus denen sie entnommen wurden. Die anderen beiden Durchläufe erfolgen in der gleichen Weise. Bei dem Verfahren werden nur die Speicherplätze benutzt, die die anfänglichen Daten enthielten, daher bezeichnet man dies auch mit „In Place".

Zentraler Teil des Am29540 ist ein durch einen 2-Bit-Mikrocode gesteuerter 19-Bit-Zähler (*Bild 4*). Mit die-

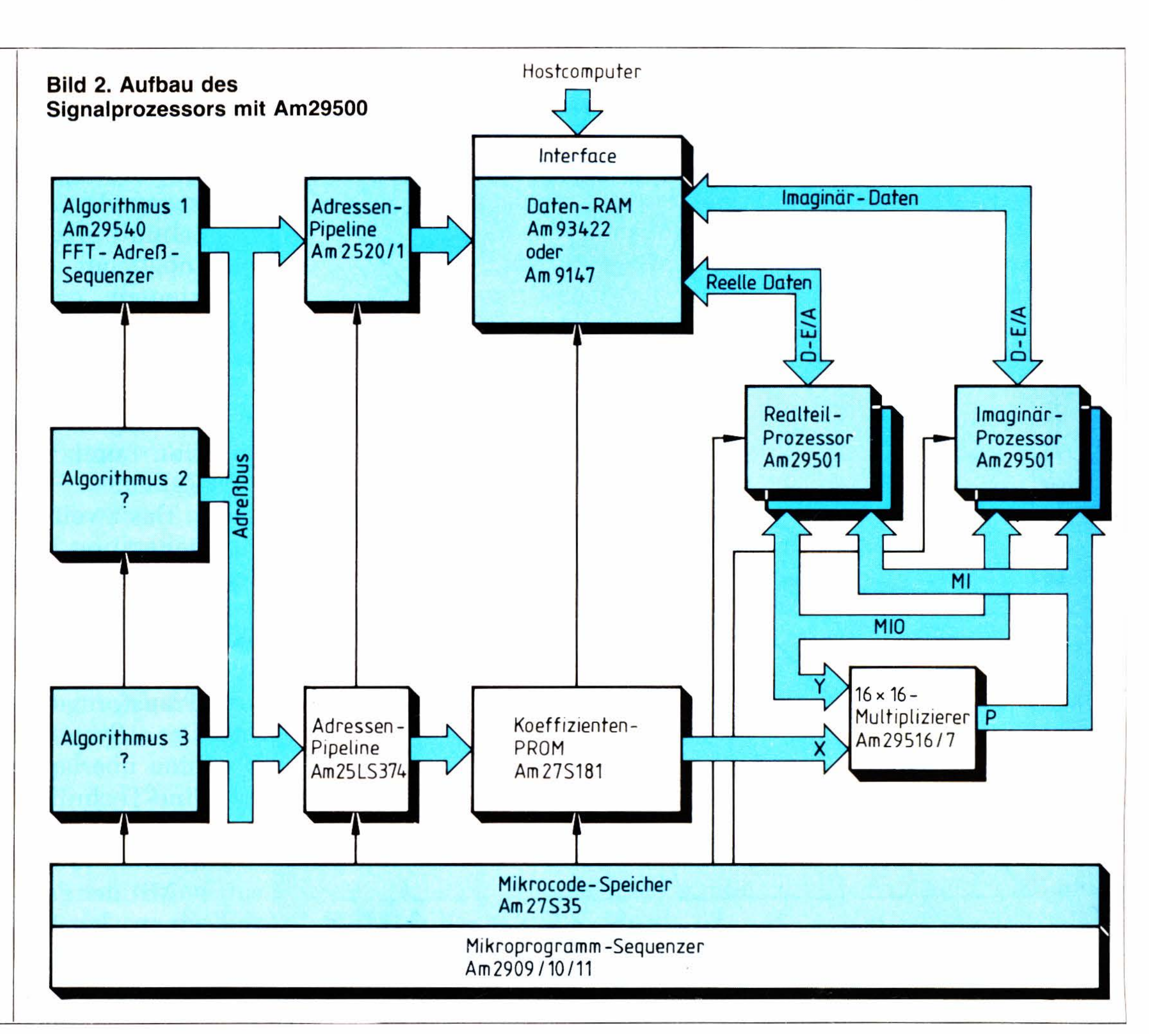


Bernard J. New wurde in London geboren. 1972 schloß er das Studium der Elektrotechnik an der Universität von Birmingham ab. Nach sechs Jahren Tätigkeit bei Plessey Marine ging er in die USA und wurde Mitarbeiter von AMD. Heute ist er Applikations Manager für digitale Signalverarbeitungssysteme. Seine Hobbys sind unter anderem Fotografieren und Schleifen von Edelsteinen.

*

Udo Renz wurde 1954 in Albstadt geboren. Nach seinem Elektronik-Studium an der FH Furtwangen arbeitete er bei der Firma Diehl (Nürnberg) als Entwicklungsingenieur, und ist seit fast drei Jahren bei AMD als Applikationsingenieur tätig. Hobbys: Segeln, Skifahren

Bild 2. Aufbau des Signalprozessors mit Am29500



sem werden die Butterfly-Sequenzen gesteuert. Es sind Transformationen von bis zu 2^{16} Punkten möglich (16 Durchgänge mit 2^{15} Butterflies). Die vier Instruktionen sind:

1. Count (Zählen) (nächste Butterfly)
2. Reset (Start der Transformation)
3. Reset Load (Offset)
4. Hold

Die Länge des Zählers ist programmierbar, so daß alle binären Transformationslängen zwischen 2 und 64 K Punkten zu verarbeiten sind. Dazu ist der Adreßbus 16 Bit breit. Bei Transformation von weniger als 64 K Punkten sind die Ausgänge der oberen Bits inaktiv. So benötigt man beispielsweise für eine 1024-Punkte-FFT lediglich 10 Adreßbits. Man kann den Am29540 auch mit einem Offset laden, der über die ungenutzten Ausgänge der oberen Bits während der Transformation ausgegeben wird. In diesem Beispiel wären das 6 Bit.

Diese Bits werden über den Adressenausgang geladen, der bidirektional ausgelegt ist. Der Vorgang erfolgt während des Starts der Transformation, indem dem Zähler das Kommando „Reset/Load" gegeben wird. Wenn kein Offset erforderlich ist, kann man das einfache Reset-Kommando verwenden. In beiden Fällen wird der Zähler auf die erste Butterfly-Operation der Transformation zurückgesetzt.

Die restliche Schaltung besteht aus kombinatorischer Logik, mit der aus den Zählerausgängen die erforderlichen Adressen und Flags decodiert werden. Wenn erforderlich, können die vorhergehenden Adressen der Butterfly-Operationen wiederholt werden. Eine solche Technik bietet sich für langsamere preiswerte Systeme an, in denen die Leseadressen der Daten als Schreibadressen an der gleichen Stelle („In Place") regeneriert werden, nachdem die Arithmetik-Operation ausgeführt worden ist. Der Butterfly-Zähler erhält erst dann den nächsten Taktimpuls, wenn die Arithmetik-Operation vollständig ausgeführt ist. Weil es nicht sinnvoll ist, das Taktsignal direkt extern durch ein Gatter zu steuern, ist die Hold-Instruktion besonders wichtig. Diese dient gleichzeitig zur Erzielung der Synchronität bei Butterfly-Operationen mit mehreren Zyklen.

Zur Festlegung der FFT-Variation sind drei Leitungen vorgesehen. Die erste ermöglicht die Auswahl zwischen DIT- (Decimation in Time) und DIF-Algorithmus (Decimation in Frequency). Man erhält mit beiden Verfahren

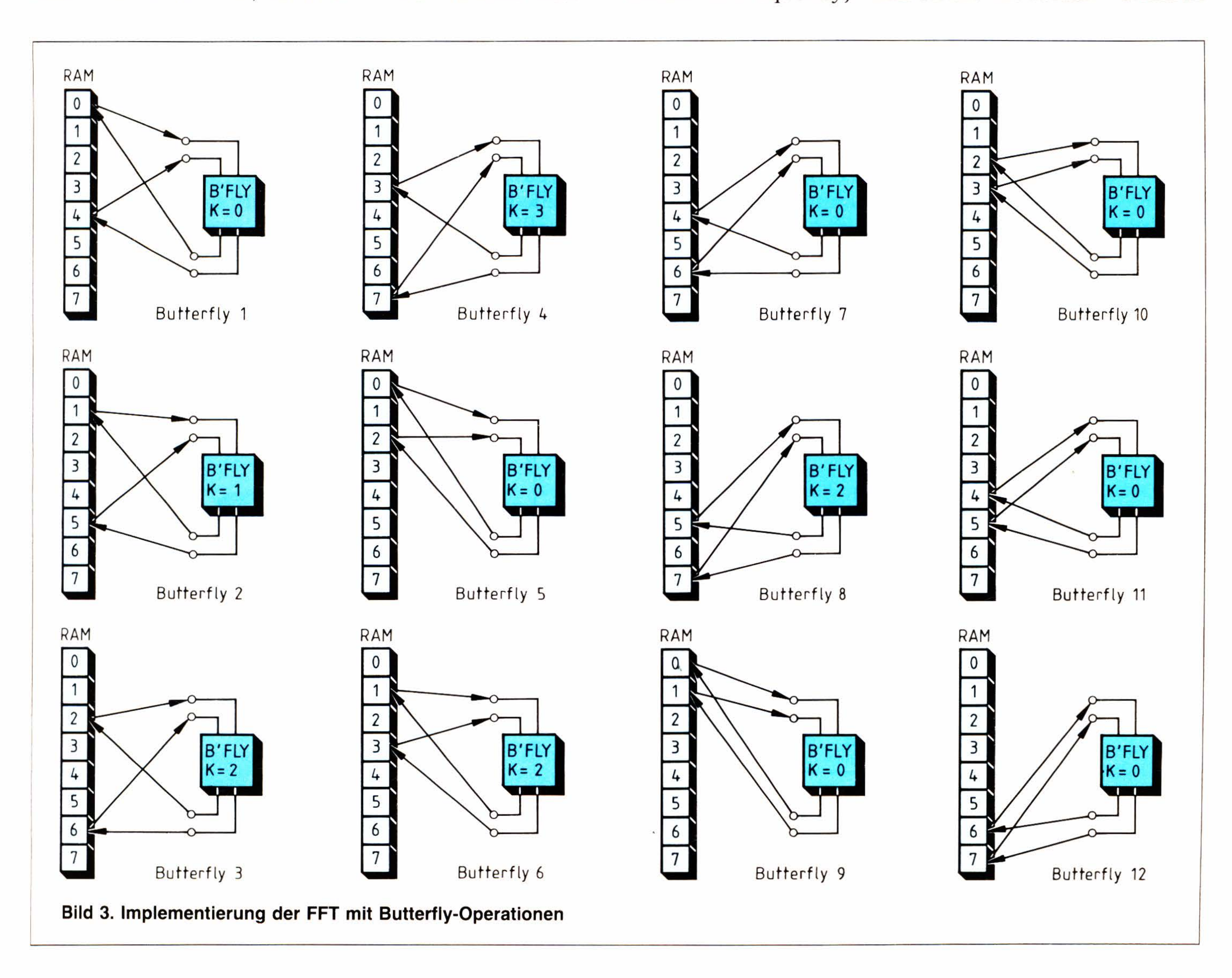


Bild 3. Implementierung der FFT mit Butterfly-Operationen

die gleichen Ergebnisse. Die weiter oben angegebene Butterfly-Formel ist ein DIF-Algorithmus, die DIT-Beziehung sieht ähnlich aus. Der wichtigste Unterschied, was den FFT-Adreßsequenzer angeht, liegt bei den komplexen Koeffizienten.

In-Place-Transformationen sind sehr speichereffizient, haben allerdings auch einen Nachteil. Wenn die zu transformierenden Daten eine natürlich geordnete Menge von Abtaststellen des Signals darstellen, weist die Ausgangsfunktion nicht die gleiche Reihenfolge auf. Die Werte sind „bitverkehrt" angeordnet. Dabei werden die Adreßbits vertauscht. Beispielsweise kann der Abtastwert von 1011 unter der Adresse 1101 gefunden werden. Wenn am Ausgang das Signal in einer bestimmten Reihenfolge vorliegen muß, kann man die Eingangssignale vorher entsprechend vertauschen. Die Adressenreihenfolge ist bei diesen zwei Möglichkeiten für Daten und Koeffizienten verschieden. Der Baustein Am29540 kann die Daten weder vertauschen noch neu ordnen, ermöglicht aber, dies vorher auszuwählen. Dazu dient das Steuersignal „PSD" (Pre-Scramled Data Control).

Die dritte Steuerleitung legt fest, ob eine Wurzel-2- oder Wurzel-4-Transformation ausgeführt wird. Die weiter oben beschriebene Butterfly-Operation mit zwei Eingängen ist eine Wurzel-2-Transformation. Bei der Wurzel-4-Transformation hat jede Butterfly-Operation vier Eingangswerte und drei komplexe Koeffizienten. Der Vorteil des Wurzel-4-Verfahrens ist, daß mehr Hardware parallel benutzt werden kann, obwohl die Operationen komplizierter sind. Der Aufwand verringert sich auf ein Viertel, nur die Hälfte der Speicherzugriffe muß ausgeführt werden. Nachteilig ist, daß die Transformationslänge auf Potenzen von vier beschränkt sind. Für DIT oder DIF, In-Place-Operationen und Vertauschen der Daten gelten dieselben Gesichtspunkte. Allerdings sind die Datenadressen stellenweise vertauscht, die Bits haben paarweise ihre Reihenfolge verändert, innerhalb des Paares wird sie beibehalten. Für die drei Leitungen zur Steuerung des FFT-Typs und für die vier Leitungen, die die Transformationslänge festlegen, sind Latches vorgesehen.

Vier Auswahleingänge ermöglichen die Benutzung von 16 unterschiedlichen Adressen, obwohl normalerweise nur eine Untermenge davon Verwendung findet. Die erste Gruppe von vier Adressen sind die Datenadressen für die Wurzel-4-Transformationen. Bei der Wurzel-2-Operation werden die Datenadressen aus den ersten zwei Auswahlcodes dieser Gruppe genommen. Die zweite Gruppe von vier Adressen sind Ausweichadressen, die bei Transformationen benutzt werden, die nicht am selben Speicherplatz erfolgen („Non-In-Place"). Dabei entsteht keine Vertauschung der Bits, allerdings verdoppelt sich der erforderliche Speicherbereich. Wenn beide Speicherhälften parallel arbeiten, sind gleichzeitig Schreib- und Leseoperationen möglich.

Mit der dritten Gruppe der Adressen kann man auf die komplexen Koeffizienten zugreifen. Für die Wurzel-4-Operation werden alle drei, für die Wurzel-2-Operation nur die erste Adresse benötigt. Diese Adressen sind mit Absicht auf den höherwertigen Stellen untergebracht, weil die gleiche Tabelle für verschiedene Transformationsgrößen Verwendung finden kann. Wenn eine Tabelle für die größte Transformation festgelegt ist, können alle anderen durch Zugriff auf eine Untermenge

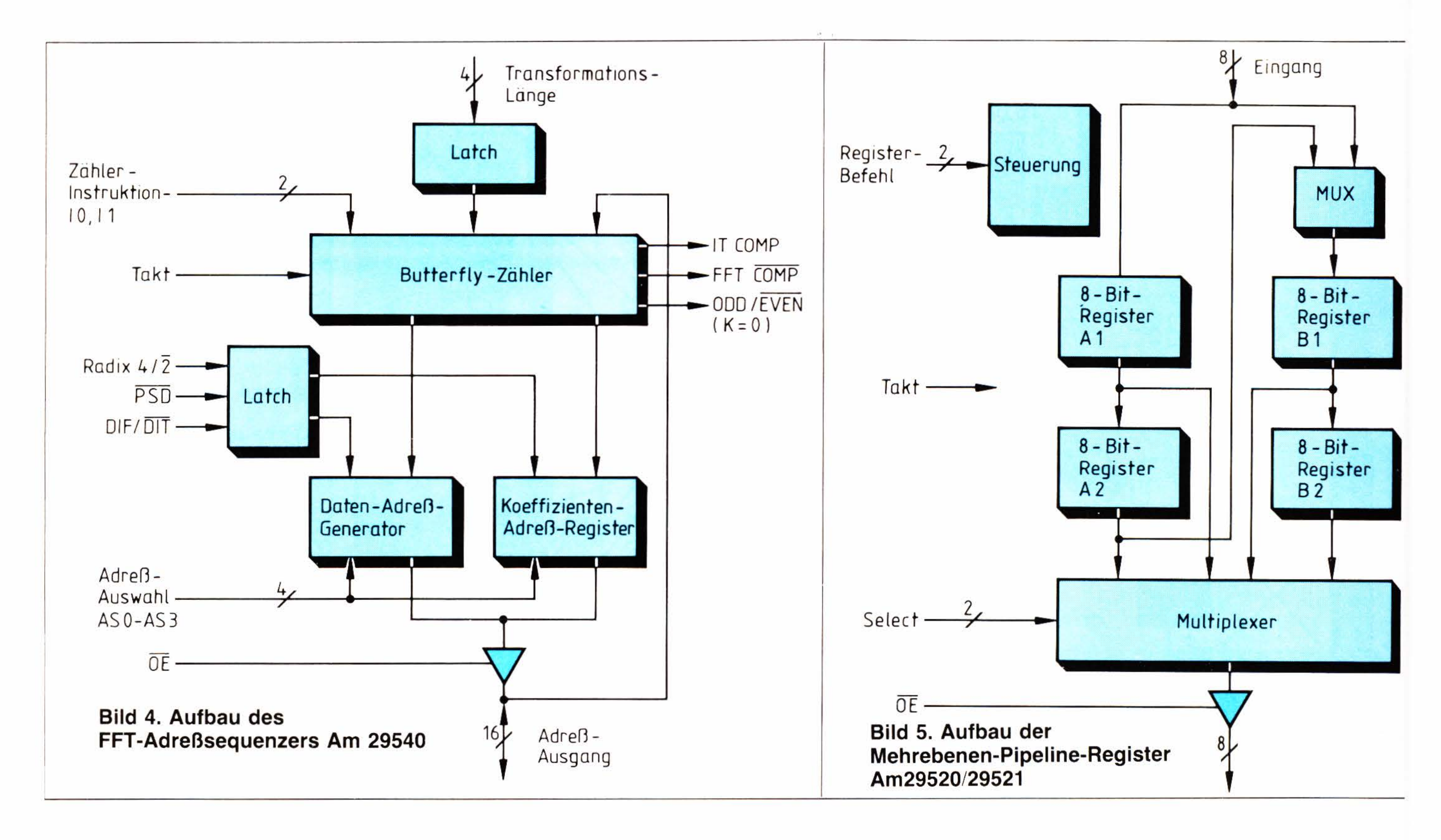


Bild 4. Aufbau des FFT-Adreßsequenzers Am 29540

Bild 5. Aufbau der Mehrebenen-Pipeline-Register Am29520/29521

daraus ausgeführt werden. Das vierte Mitglied dieser Gruppe wird zur Adressierung der Tabelle zum „Shading“ benutzt. Es handelt sich dabei um eine Multiplikation der Eingangsdaten mit einer Wichtungsfunktion. Die Ausgangswerte der Transformation werden dadurch modifiziert, wenn z. B. Frequenzanteile gedämpft sind, die nicht genau mit den abgetasteten Stellen übereinstimmen.

Die letzte Gruppe aus vier Adressen dienen der Transformation reeller Werte (RVI – Real Valued Input). Damit wird das Spektrum reeller Signale ermittelt. Der Speicherbedarf ist dabei auf die Hälfte reduziert und der Imaginärteil des Eingangssignals muß nicht auf Null gesetzt werden.

Drei Flags vereinfachen den Betrieb des Mikroprogramm-Sequenzers. „FFT COMPLETE“ zeigt an, daß die letzte Butterfly-Operation der Transformation erreicht ist. Ein Schleifenzähler ist daher überflüssig. Das Mikroprogramm muß nur Butterfly-Operationen ausführen, bis dieses Flag aktiviert ist; das Programm schließt bei jeder beliebigen Transformationslänge beim richtigen Punkt ab. Ein Durchgang bedeutet einen maximalen Wortzuwachs um 9 dB. Wenn im oberen Bereich zwei Bit verarbeitet werden, kann kein Überlauf auftreten. Bei Block-Fließkomma wird ein einzelner Exponent für die zu transformierenden Daten benutzt. Die Überprüfung der Schreibdaten erfolgt während eines vollständigen Durchlaufs. Wenn die Daten den Kopf erreichen, werden sie zur Vermeidung von Problemen geteilt. Das Flag „ITERATION COMPLETE“ erscheint während der letzten Butterfly-Operation jedes Durchgangs, um diesen Vorgang zu steuern.

Das dritte Flag hat zwei Funktionen. Normalerweise unterscheidet es, ob die gerade oder ungerade Spalte der Butterflies verarbeitet wird. Gesteuert wird damit die Umschaltung zwischen Schreib- und Lesespeicher bei Operationen, die nicht auf denselben Plätzen stattfinden. Die maximale Taktrate beträgt beim Zugriff 10 MHz, zwischen Adressenanwahl und Ausgabe des Signals vergehen 50 ns.

5 Betrieb des Am29540

Weil der Baustein ausschließlich auf dem Adreßbus arbeitet, kann der FFT-Adreßsequenzer mit jedem Arithmetikprozessor betrieben werden. Das kann z. B. eine beliebige MOS-CPU mit Fließkomma-Coprozessor sein. Der Sequenzer übernimmt dabei die Adreßberechnung. Sinnvoller ist die Kombination mit einem schnellen bipolaren, mikroprogrammierten System (Bild 2), das alle vier Zyklen einer Butterfly-Operation durchführen kann. Dafür sind drei Adressen erforderlich, daher wird der Am29540 synchron mit Leerzyklus betrieben.

Um hohen Durchsatz erreichen zu können, müssen die Butterflies überlappend und mit der Pipeline-Methode ausgeführt werden. Es ist daher erforderlich, die Datenadressen während der arithmetischen Operation (oder anderen Speicheroperationen) zu speichern. Die Mehrebenen-Pipeline-Register Am29520/29521 (*Bild 5*) ermöglichen den Aufbau einer Adreßpipeline parallel zur Datenpipeline. Zwei Adressen werden erzeugt und in die Pipeline geladen. Die Koeffizienten-Adressen lädt man in das einfache Adreßregister Am2954. Alle vier Zyklen wird der Butterfly-Zähler geladen, wobei während der anderen drei Zyklen ein Hold-Befehl gegeben wird. Der Adreßausgang ist während einer der vier Zyklen inaktiv. In sehr schnellen Systemen können auch mehrere Bausteine vom Typ Am29540 parallel arbeiten, um die Adressen in der notwendigen Geschwindigkeit zu erzeugen.

6 Ausführung der schnellen Faltung

Wie bereits beschrieben, hat die schnelle Faltung im Frequenzbereich drei Phasen: Transformation, Multiplikation und Rücktransformation. Der Hostcomputer lädt die Eingangsdaten in das RAM. Danach wird per Makro-Instruktion die Vorwärts-FFT ausgelöst, wobei die Bits vertauscht sind. Darauf erfolgt die Multiplikation mit der im PROM gespeicherten Übertragungsfunktion. Wenn diese Funktion dynamisch geändert werden muß, kann das PROM auch durch ein RAM ersetzt werden, das der Hostcomputer lädt.

Um an die richtige Reihenfolge angepaßt zu sein, muß die Übertragungsfunktion auch bitweise vertauscht gespeichert sein. Weil die Reihenfolge, in der die Multiplikationen ausgeführt werden, unkritisch ist, kann ein einzelner FFT-Durchlauf auf alle Datenpunkte gleichzeitig zugreifen. Die Butterfly-Operation wird durch die Multiplikation mit den entsprechenden Punkten der Übertragungsfunktion ersetzt. In einem umfangreichen Prozessor kann man einen der nicht verwendeten Adreßgeneratoren für diese Operation benutzen. Dadurch vermeidet man, daß Daten und Übertragungsfunktion unter einer Adresse zu speichern sind. Ein solcher Prozessor kann aus den bipolaren Bit-Slice-Elementen Am2901 oder dem 16-Bit-Controller Am29116 aufgebaut werden.

Der einzige Unterschied bei der Ausführung der Rücktransformation ist, daß die komplexen Koeffizienten hinzugefügt werden müssen. Dieses erreicht man durch eine einfache Veränderung des Mikrocodes für den Arithmetikprozessor, damit Additionen und Subtraktionen, wo erforderlich, ausgetauscht werden. Wenn die Eingangsdaten bitweise vertauscht vorliegen, muß die entsprechende inverse Transformation gewählt werden. Das Augangssignal liegt in natürlicher Reihenfolge vor, so daß sich Vertauschungen erübrigen. Die Korrektur für die zyklische Faltung nimmt der Hostrechner vor.

Literatur

[1] Designer's Guide to Digital Signal Processing; parts 1-5, TRW LSI Products. EDN 1981, March 18 – May 13.
[2] *Rabiner, Gold:* Theory and Applications of Digital Signal Processing. Pretice-Hall 1975.
[3] *Brigham, E. O.:* Fast Fourier Tranform. Pretice-Hall 1974.

Dipl.-Phys. Martin Winterer

Signalprozessor löst rechenintensive Probleme

Bereits in den 40er Jahren erkannte man die Vorteile einer digitalen Signalverarbeitung. Sie blieb allerdings Spekulation, da die dazu benötigte Technik noch fehlte. Erst die sich nach 1950 entwickelnde Computerindustrie legte den Grundstein einer „digitalen Revolution": Mit der Einführung integrierter Schaltungen und weitergehender Miniaturisierung führte sie schließlich zu den ersten Mikroprozessoren, die ungeheuere Auswirkungen auf alle Bereiche der Technik hatten. Parallel dazu gewann auch die digitale Verarbeitung von ursprünglich analog vorliegenden Signalen an Bedeutung. Zu Anfang hatte man es mit relativ teuren und umfangreichen Systemen zu tun. Doch auch hier zeichnet sich ein Trend zu programmierbaren und damit in großen Stückzahlen herstellbaren Bausteinen ab, ähnlich der Entwicklung, die zu den Mikroprozessoren führte. Die heutige Technik dieser digitalen Signalprozessoren (DSP) erlaubt eine sinnvolle Anwendung für Signalfrequenzen bis über 100 kHz hinaus.

Anstoß für diese Arbeit gab der von der Firma ITT Intermetall kürzlich vorgestellte Signalprozessor mit der Typenbezeichnung UDPI 01. Eine Zusammenstellung der Eigenschaften zeigt *Tabelle 1*. Der Grundgedanke bei der Entwicklung seiner Prozessorarchitektur war nicht nur, einen Mikroprozessor mit zusätzlichem Hardware-Multiplizierer herzustellen, sondern einen Chip, der die speziellen Aufgaben, die an ein digitales signalverarbeitendes System gestellt werden, optimal erfüllt. Neben einer Beschreibung dieses neuen Bausteins ist die Absicht dieses Artikels vorrangig darin zu sehen, die angesprochenen Aufgaben und daraus erwachsende Forderungen darzulegen sowie Applikationen aufzuzeigen.

1 Methoden der digitalen Signalverarbeitung

Ein digitales System benötigt als Eingangsdaten eine Folge von Zahlenwerten, die der gewünschten Funktion entsprechend verschiedenen mathematischen Operationen unterworfen wird. Das Endprodukt dieser Verknüpfungen bildet wieder eine Folge von Zahlen als Ausgangswerte. In analogen Systemen erhält man die Eingangszahlenfolge durch Abtasten eines analogen Signals mit anschließender Analog/Digital-Umsetzung. Durch diese Aktionen wird das Signal sowohl zeit- als auch amplitudenquantisiert. Aufgrund der Zeitquantisierung muß man ein bandbegrenztes Signal voraussetzen, da sonst Aliasing-Effekte auftreten. Als Auswirkung der Amplitudenquantisierung entsteht ein Fehler, der sich als Rauschen im Signal äußert und den man als Quantisierungsfehler bezeichnet. Diese beiden Phänomene sind in der betreffenden Literatur eingehend untersucht worden, deshalb soll hier darauf nicht näher eingegan-

Tabelle 1. Steckbrief des Signalprozessors UDPI 01

- Befehlszykluszeit 100 ns
- Festkomma-Zweierkomplement-Arithmetik
- 16 × 16 → 31-Bit-Multiplikation in 200 ns (inkl. Addieren, Operanden holen, Ergebnis abtransportieren)
- Zwei Akkumulatoren mit 31 Bit
- Wortlänge für Daten und Befehle: 16 Bit
- Interner Speicherbereich:
  - Programm-ROM: 1 K × 16 Bit
  - Daten-ROM: 72 × 16 Bit
  - Daten-RAM: 440 × 16 Bit
- Vierfacher Unterprogramm-Stack
- Mehrere Ein-/Ausgabemöglichkeiten:
  - Seriell (schnell, bis zu 5 MBit/s)
  - Seriell (langsam, IM-Bus)
  - Parallel (16 Bit)
- Kompatibel mit 16-Bit-Mikroprozessoren, wie z. B. MC 68000
- Versorgungsspannung 5 V, Leistungsaufnahme 80 mW
- 2,4 μm Silicon-Gate-CMOS-Technologie
- Emulator-IC mit externem Programmspeicher verfügbar
- Cross-Assembler und Softwaresimulator geschrieben in FORTRAN (Standard 77 mit einigen wenigen Zusatzfunktionen, die auf DEC-Rechner verfügbar sind)

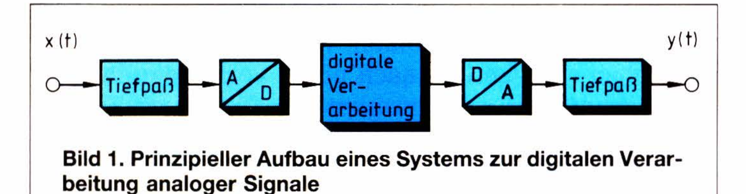


Bild 1. Prinzipieller Aufbau eines Systems zur digitalen Verarbeitung analoger Signale

gen werden [1...4]. Andererseits wird aus der Ausgangszahlenfolge durch eine Digital/Analog-Umsetzung ein analoges Signal hergestellt, das zunächst als Treppenkurve vorliegt. Durch Glättung mit einem Tiefpaßfilter, das alle Frequenzen herausfiltert, die von der Stufung herrühren, erhält man das gewünschte analoge Ausgangssignal. *Bild 1* zeigt schematisch die Funktionsblöcke eines kompletten digitalen Systems zur Verarbeitung analoger Signale.

2 Aufgaben und Forderungen

Die Funktionen, die ein digitales System ausführen kann, sind vielfältiger Natur; der Großteil der Aufgaben wird jedoch mit Hilfe digitaler Filterung bzw. verwandter Algorithmen beschrieben.

Die Grundoperation aller digitalen Filter (und vieler weiterer Anwendungen) ist die Multiplikation mit anschließender Addition. Um dies zu verdeutlichen, folgt hier ein kurzer Blick auf die Struktur eines digitalen Filters. Man unterscheidet zwischen transversalen (bzw. nichtrekursiven) und rekursiven Filtertypen (*Bild 2*).

In transversalen Filtern (Bild 2*a*) wird die Summe der N letzten Abtastwerte gebildet, jeweils gewichtet mit einem bestimmten Koeffizienten. Für einen einzelnen Schritt benötigt man dazu eine Operation in der Form:

$$Y(i) = Y(i) + Bj * X(i-j)$$

Der Abtastwert X(i–j) ist mit dem Koeffizienten Bj zu multiplizieren und das Ergebnis zu Y(i) zu addieren. Signalprozessoren sind dazu eingerichtet, diese Operation mit maximalem Durchsatz verarbeiten zu können.

Zusätzlich muß bei jedem Abtasttakt ein neuer Wert gespeichert und alle vorherigen Abtastwerte um einen Takt verzögert werden (oft als z^{-1}-Funktion bezeichnet). Der jeweils älteste fällt dabei weg. Zur Realisierung dieser Funktion gibt es zwei verschiedene Ansätze: Zum einen wird in manchem DSP diese Umspeicherung tatsächlich vorgenommen, d.h. ein Datenwort wird über einen Datenbus an eine andere Stelle bewegt. Diese zusätzliche Belastung des Datenbusses verbietet sich aber in bestimmten Fällen – doch dazu später. Zum anderen läßt man sich von dem Gedanken leiten, daß man nicht die Daten auf andere Adressen zu schieben braucht, sondern lediglich die Zuordnung der Adressen zu den Daten ändert. Dies führt zu einer neuen Art von Adressiertechnik, der Modulo-Adressierung.

Wie diese funktioniert, verdeutlicht *Bild 3*. Es zeigt die Ablage der Abtastwerte (z.B. eines transversalen Filters vierter Ordnung) im Speicher. Der aktuell adressierte Datenwert (zu Beginn eine leere Speicherzelle) ist durch einen Pfeil gekennzeichnet (a). Auf diese Stelle wird nun der neue Abtastwert eingelesen und gleichzeitig der Datenzeiger um eins dekrementiert. Die früheren

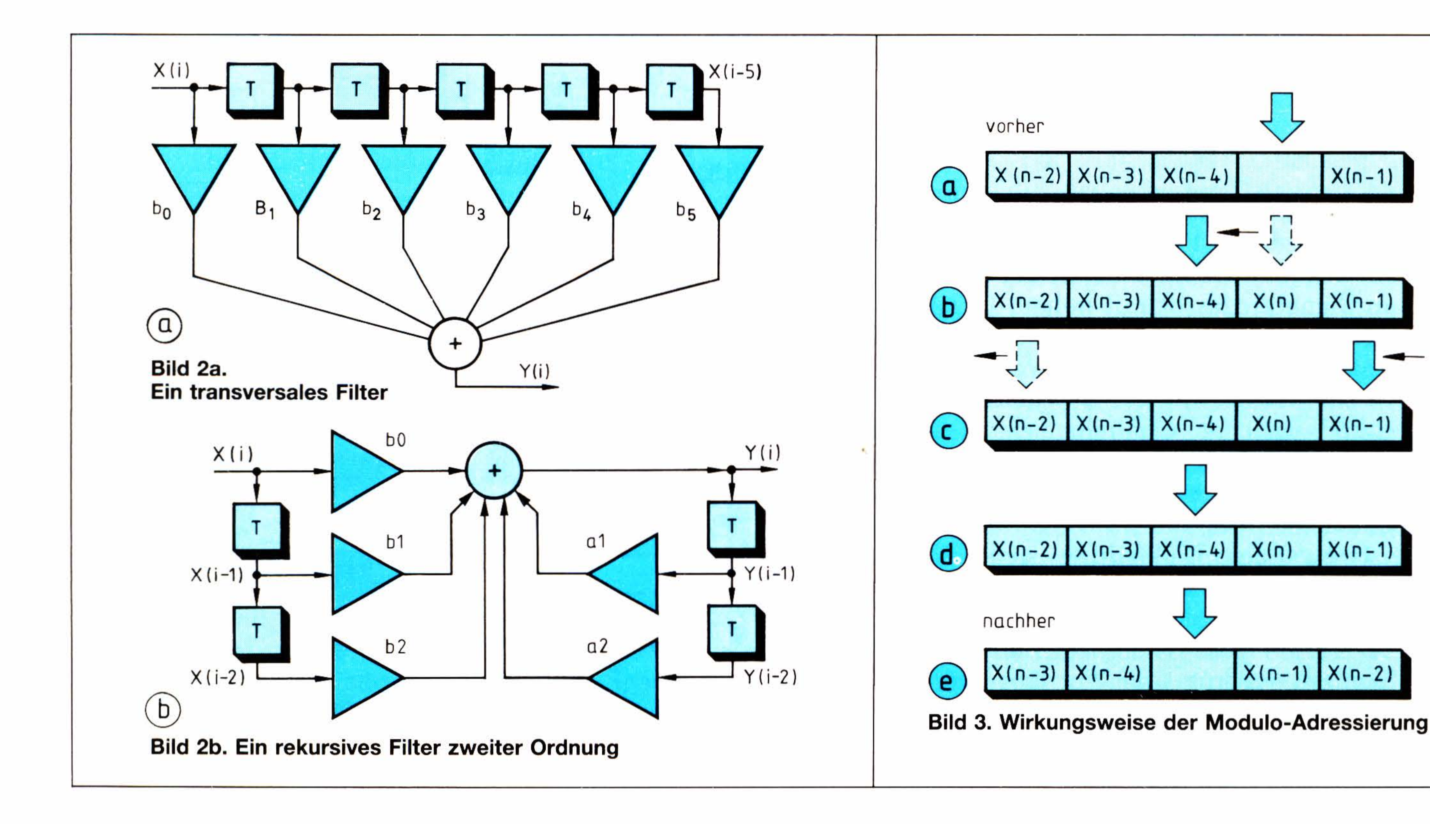


Bild 2a. Ein transversales Filter

Bild 2b. Ein rekursives Filter zweiter Ordnung

Bild 3. Wirkungsweise der Modulo-Adressierung

Abtastwerte sind so angeordnet, daß der Zeiger nun den ältesten Wert beschreibt (b). Dieser Wert wird gelesen, mit dem Koeffizienten B_4 multipliziert, in einem Akkumulator abgespeichert und der Datenzeiger um eins dekrementiert. Erneut wird der betreffende Wert gelesen, mit einem Koeffizienten multipliziert und zu dem Akkumulatorinhalt addiert. Interessant ist der Fall (c). Der Adreßzähler unterschreitet beim Dekrementieren den Wert null. Dann sorgt eine Logik dafür, den Datenzeiger auf der Stelle vier aufzusetzen, d.h. der Inhalt des Zeigers wird hier Modulo 5 verstanden. Sind alle fünf Abtastwerte verarbeitet, befindet sich das System im Zustand (d). Der Datenzeiger steht eine Stelle tiefer als zu Beginn (er wurde sechsmal dekrementiert: einmal beim Schreiben des neuen Abtastwertes, fünfmal beim Verarbeiten der fünf Werte). Die aktuelle Abtastperiode ist beendet, das Programm erwartet einen neuen Abtastwert. Der eben noch eingelesene Wert ist nun nicht mehr der neueste, sondern um einen Takt gealtert, der bisher älteste wird nicht mehr berücksichtigt. Auf dessen Stelle zeigt aber der Datenzeiger, d.h., er zeigt wie zu Beginn auf eine freie Stelle. Benennt man alle Werte neu, erhält man die Darstellung (e). Hier wird deutlich, daß die Endstellung der Anfangsstellung entspricht, wenn man sich nur für die Datenwerte interessiert, die der Zeiger adressiert.

Will man mit einem Signalprozessor längere digitale Filter verarbeiten, sollte er die Möglichkeit dieser Modulo-Adressierung besitzen. Wichtig ist auch, daß man die Basis der Modulo-Operation (Länge des Ringspeichers) frei wählen kann.

Bei der oben erwähnten Grundoperation werden für jeden Zyklus zwei Werte über einen Datenbus dem Multiplizierer zugeführt. In vielen Anwendungen (z.B. Korrelationen und Polynome) kommt jedoch eine allgemeinere Form der „Multipliziere und Addiere“-Operation vor:

$$A(i) = B(i) + C(i) * D(i)$$

Hier müssen innerhalb eines Zyklus vier Datenworte heran- bzw. wegbewegt werden. Auch diese allgemeine Operation mit ihrer hohen Auslastung des Datenbusses sollte ein Signalprozessor mit hohem Durchsatz verarbeiten können. Aus diesem Grund verbietet sich im allgemeinen eine zusätzliche Bewegung über einen Datenbus zur Ausführung der z^{-1}-Funktion.

Ein grundsätzlicher Qualitätsmaßstab für digitale Systeme ist die erzielbare Rechengenauigkeit. Es hat sich gezeigt, daß selbst für HiFi-Anwendungen eine Festkommadarstellung mit 16 Bit Auflösung ausreicht. Die meistverwendete Darstellung ist die Zweier-Komplement-Darstellung (ZDK) mit linksjustiertem Komma (auf eins normierte Zahlen). Geht man von 16 Bit Auflösung aus (15 Bit plus 1 Bit Vorzeichen), so hat ein Multiplikationsergebnis eine Wortlänge von 31 Bit (30 Bit plus 1 Bit Vorzeichen). Ist das Rechenwerk in der Lage, diese 31-Bit-Worte aufzunehmen, begeht man auch bei fortlaufenden Akkumulationen keinen Rechenfehler. Es dürfen sogar beliebig viele Überläufe (Ergebnis <–1 oder => +1) stattfinden, wenn für jeden Überlauf nach oben ein Überlauf nach unten vorhanden ist (d.h. das Ergebnis auf eins normiert ist). Der Ausgangswert besitzt dann immer noch 31 Bit Genauigkeit und wird z.B. im Filter Bild 2a erst nach erfolgter Akkumulation einer Quantisierung auf 16 Bit unterzogen.

Anders liegt der Fall bei rekursiven Strukturen (Bild 2b). Hier werden die Ausgangswerte zum Eingang zurückgekoppelt. Dies hat eine wichtige Konsequenz für die Rechengenauigkeit: Aus einer 16×16-Bit-Multiplikation, deren Ausgangswert 31 signifikante Stellen einnimmt, können nur 16 Bit weiterverwendet werden. Dieser Vorgang stellt wieder eine Quantisierung dar und führt somit zu einem Quantisierungsrauschen, das auch zum Eingang zurückgeführt wird. Daneben treten in rekursiven Filtern weitere Probleme auf, wie z.B. Stabilitätsprobleme, Grenzzyklen und Überlaufsschwingungen. Zur Beeinflussung von Grenzzyklen sollte der Prozessor zur Quantisierung wahlweise Abschneiden oder Runden vorsehen sowie eine anschaltbare Sättigungscharakteristik (Klemmung) besitzen, die zur Verhinderung von Überlaufsschwingungen dient.

Aufgrund dieser Überlegungen kann man folgende Forderung an einen Signalprozessor stellen:

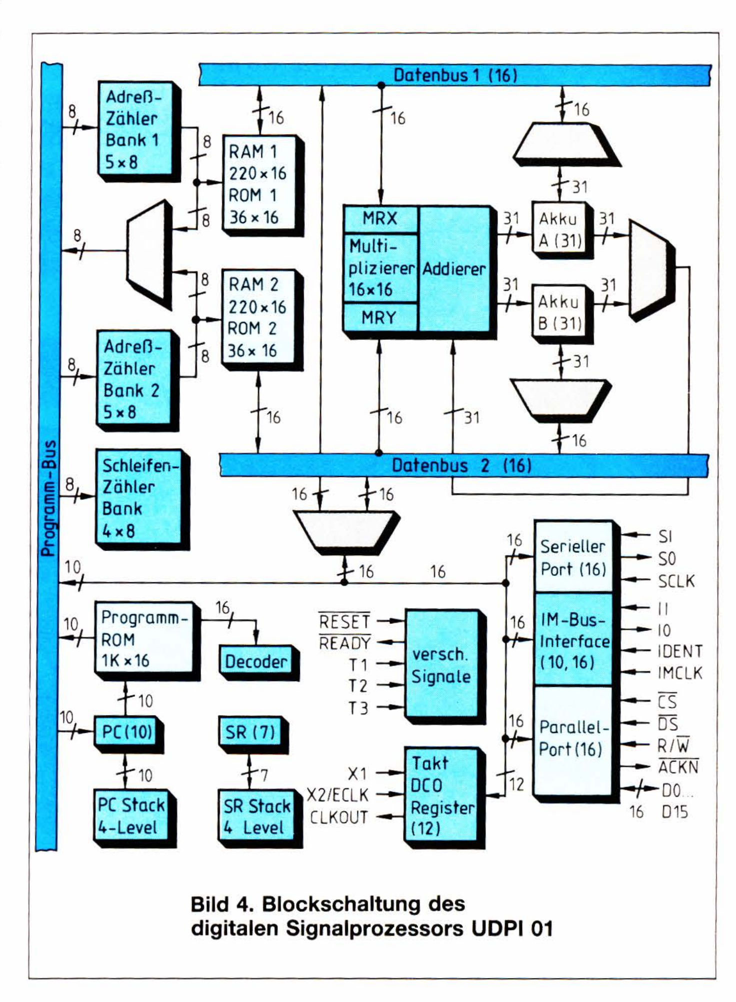


Bild 4. Blockschaltung des digitalen Signalprozessors UDPI 01

Daten	16 Bit
Multiplizierer	16×16 → 31 Bit
Akkumulator	31 Bit
Auswählbare Quantisierung:	Runden oder Abschneiden
anschaltbare Klemmung	
allgem. Grundoperation: mit maximalem Durchsatz	$A(i) = B(i) + C(i) \times D(i)$
automatische z^{-1}-Funktion	

3 Architektur des UDPI 01

Dieser Baustein ist aufgrund seiner Architektur in der Lage, die oben angesprochenen Anforderungen zu erfüllen. Er benützt eine für hohe Geschwindigkeit vorteilhafte „Harvard-Struktur" (*Bild 4*). Im Gegensatz zur üblichen „Von-Neumann-Architektur" liegen Daten- und Programmspeicher in getrennten Bereichen. Dies ermöglicht eine parallele Verarbeitung von Instruktions-Fetch und Datentransport.

Eine direkte Fortführung dieses Gedankens ist die weitere Aufteilung des Datenbusses in zwei gleichwertige Teile, da ein Multiplizierbaustein gleichzeitig zwei Multiplikatoren benötigt, die ihm damit parallel zugeführt werden können. Obwohl das Hauptmerkmal eines digitalen Signalprozessors, das ihn gegenüber einem Standardmikroprozessor auszeichnet, dieser Hardwaremultiplizierer ist, liegt der entscheidende Vorteil jedoch darin, diesem Multiplizierer auch mit maximalem Durchsatz Daten bereitzustellen, bzw. dessen Ergebnisse abzuholen und weiterzuverarbeiten. Diese Tatsache wird oft unterschätzt und allzu viel Gewicht auf die Leistung des Multiplizierers gelegt.

Betrachtet man wiederum den allgemeinen Fall einer „Multipliziere und Addiere"-Operation:

$a(i) = b(i) + c(i) \times d(i)$

Hier müssen innerhalb einer Operation vier Datenwerte bewegt werden. Legt man zugrunde, daß in einem Befehlszyklus nur ein Datum bewegt werden kann, so würde man in einem Einbussystem für diese Grundoperation vier Befehlszyklen benötigen. Verteilt man diese Datentransporte auf zwei Datenbusse, sind das immer noch zwei Datenwerte pro Bus und Operation. Die Multiplikations- und Additionseinheit hat somit zwei Zyklen Zeit, eine schnellere Recheneinheit würde keine Geschwindigkeitsvorteile mehr bieten.

Nach diesen generellen Überlegungen sollen nun die wichtigsten, in der Schemazeichnung aufgezeigten Blöcke beschrieben werden.

3.1 *Arithmetik-Einheit*

Die Arithmetik-Einheit besteht aus einem 16×16-Bit-Multiplizierer und einem 31-Bit-Addierer. Zwei Register MRX und MRY dienen zur temporären Speicherung der Multiplikatoren. Das Ergebnis bzw. den Additionswert nimmt einer der beiden 31-Bit-Akkumulatoren auf. Eine Multiplikation plus einer Addition bzw. einer Subtraktion wird in zwei Instruktionszyklen zu je 100 ns ausgeführt. Ergebnisse können unter Softwarekontrolle auf 16 Bit abgeschnitten oder gerundet werden.

Der Akkumulator A ist dem Datenbus 1, der Akkumulator B dem Datenbus 2 zugeordnet. Wahlweise kann man jeweils den High-Part (16 Bit: Bit 30 bis Bit 15) oder den Low-Part (15 Bit: Bit 14 bis Bit 0) eines Akkumulators mit dem entsprechenden Datenbus verbinden. Außerdem ist ein Transport von einem Akkumulator zum anderen möglich.

3.2 *Programmzähler und Programmspeicher*

Ein 10-Bit-Programmzähler adressiert einen Adreßraum von 1K Worten zu je 16 Bit. Der Programmspeicher ist als ROM ausgelegt und wird für eine bestimmte Aufgabe maskenprogrammiert. Eine zweite verfügbare Version des Signalprozessors (UDPI EC) adressiert statt des internen ROM einen externen Speicher.

3.3 *Adreßzähler und Datenspeicher*

Der Datenspeicher besitzt 512 Worte zu je 16 Bit insgesamt. Er ist unterteilt in zwei Blöcke mit je 220 Worten statisches RAM und 36 Worte ROM. Je einer dieser Blöcke ist mit einem der beiden Datenbusse verbunden. Zur Adressierung von Daten innerhalb der beiden Datenbänke dienen zwei Blöcke von Adreßzählern. In jedem Block stehen fünf Adreßzähler mit 8 Bit Wortlänge zur Verfügung. Zur Unterstützung einer Verarbeitung von größeren Datenblöcken können die Adreßzähler nach jedem Datentransport automatisch inkrementiert oder dekrementiert werden. Eine Besonderheit bilden die Adreßzähler 0 in jeder Bank: Unterschreitet der Inhalt dieser Zähler beim Dekrementieren den Wert Null, so wird der Inhalt des Adreßzählers 1 in den Zähler 0 kopiert. Diese Eigenschaft kann dazu verwendet werden, die Funktion eines Schieberegisters zu simulieren, ohne die gespeicherten Daten tatsächlich zu verschieben.

3.4 *Ein-/Ausgabe*

Zum Datenaustausch mit der Peripherie stehen zwei serielle und eine parallele Schnittstelle zur Verfügung. Die parallele Schnittstelle ist in erster Linie dafür gedacht, mit hoher Geschwindigkeit Daten in einem 16-Bit-Mikroprozessorsystem zu übertragen. Die Datenübergabe bedient sich eines „Handshake"-Verfahrens (68000-kompatibel) mit weitgehender Steuerung per Software, so daß auch andere Abläufe möglich sind.

Das schnelle serielle Interface vereinfacht die Programmierung periodisch ablaufender Vorgänge, wie sie für digitale Filterung typisch sind. Mit diesem Interface können serielle Daten mit einer Rate von 5 MBit/s übertragen werden. Bemerkenswert daran ist, daß innerhalb eines Übertragungszyklus bis zu sieben 16-Bit-Worte

eingelesen und bis zu vier 16-Bit-Worte ausgelesen werden können, wobei Ein- und Ausgabe parallel ablaufen.

Zusätzlich steht ein langsames serielles Interface zur Verfügung, mit dessen Hilfe ein externer Steuerprozessor auf einfachste Weise veränderliche Parameter des Signalprozessors steuern oder überwachen kann. Dieses sogenannte IM-Bus-Interface kommuniziert über vier Leitungen mit der Peripherie, wobei die Steuerfunktion auf einem µC in Software realisiert werden kann, sobald nur auf einem E/A-Port eine Eingangs- und drei Ausgangsleitungen zur Verfügung stehen.

4 Instruktionssatz

Der Befehlssatz des UDPI 01 umfaßt arithmetische Befehle, Akkumulator-, Sprung-, Steuer- und Transportbefehle.

Die Gruppe der arithmetischen Befehle umfaßt die Multiplikationsbefehle mit optionaler Addition. Es können ausgeführt werden: Multipliziere 2 Werte, addiere oder subtrahiere optional einen Akkumulatorinhalt und speichere das Ergebnis in einen Akkumulator. Zusätzlich kann ein Akkumulator aus dem Datenspeicher neu geladen und das Ergebnis in die Datenspeicher wegtransportiert werden. Bei dieser Aktion kann man zwischen Abschneiden und automatischem Runden per Flag im Status-Register wählen. Diese Instruktionen benötigen zur Ausführung 2 (bzw. 3) Zyklen, wobei im zweiten Zyklus bereits mit dem Abarbeiten der nächsten Instruktionen begonnen wird. So ist es möglich, alle vorbereitenden Aktionen wie Register-Vorladen und Ergebnis-Transport parallel zum Rechenvorgang auszuführen. Dadurch reihen sich Rechenzyklen mit sequentiellen Datenblöcken lückenlos aneinander. Die Geschwindigkeit des Rechenwerks wird so optimal genutzt. Zum besseren Verständnis betrachtet man das Beispiel in *Bild 5*.

Der Inhalt eines Akkumulators kann verschiedenen Operationen unterworfen werden. Diese sind: Absolutwertbildung, Löschung, Komplementierung, arithmetisches und logisches Schieben, Negieren, Inkrementieren, Runden auf 16 Bit, sowie der Transport von einem zum anderen Akkumulator mit gleichzeitiger Ausführung einiger der aufgeführten Operationen.

Die Sprungbefehle umfassen unbedingte und bedingte Sprünge, Schleifenkontrollinstruktionen, sowie Instruktionen zur Unterprogrammtechnik. Die letzte Gruppe umfaßt Instruktionen, die Datenbewegungen zwischen den verschiedenen Funktionsblöcken veranlassen.

5 Beispielprogramme

5.1 *Rekursives Filter zweiter Ordnung*

Zur besseren Anschauung soll hier ein kurzes Filterbeispiel explizit programmiert werden. Das Netzwerk des gewünschten Filters ist dargestellt in Bild 2b. Hier müssen fünf Multiplikationen ausgeführt werden, vier davon mit anschließender Addition. Dazu werden die Instruktionen MUL und MAA verwendet (*Tabelle 2*). Doch nun zum Programmablauf: Zu Beginn (nach einem Reset oder dem Systemstart) werden zunächst die Adreßzähler in Ordnung gebracht, indem sie mit einem definierten Wert geladen werden. Durch die Instruktion MOV #AC11,4 richtet man sich einen Ringspeicher mit der Länge fünf ein. Man kann diesen Speicherbereich noch nullsetzen, um einen definierten Ausgangszustand herzustellen. Das Programm wartet danach in einer Endlosschleife auf das Eintreffen von Abtastdaten

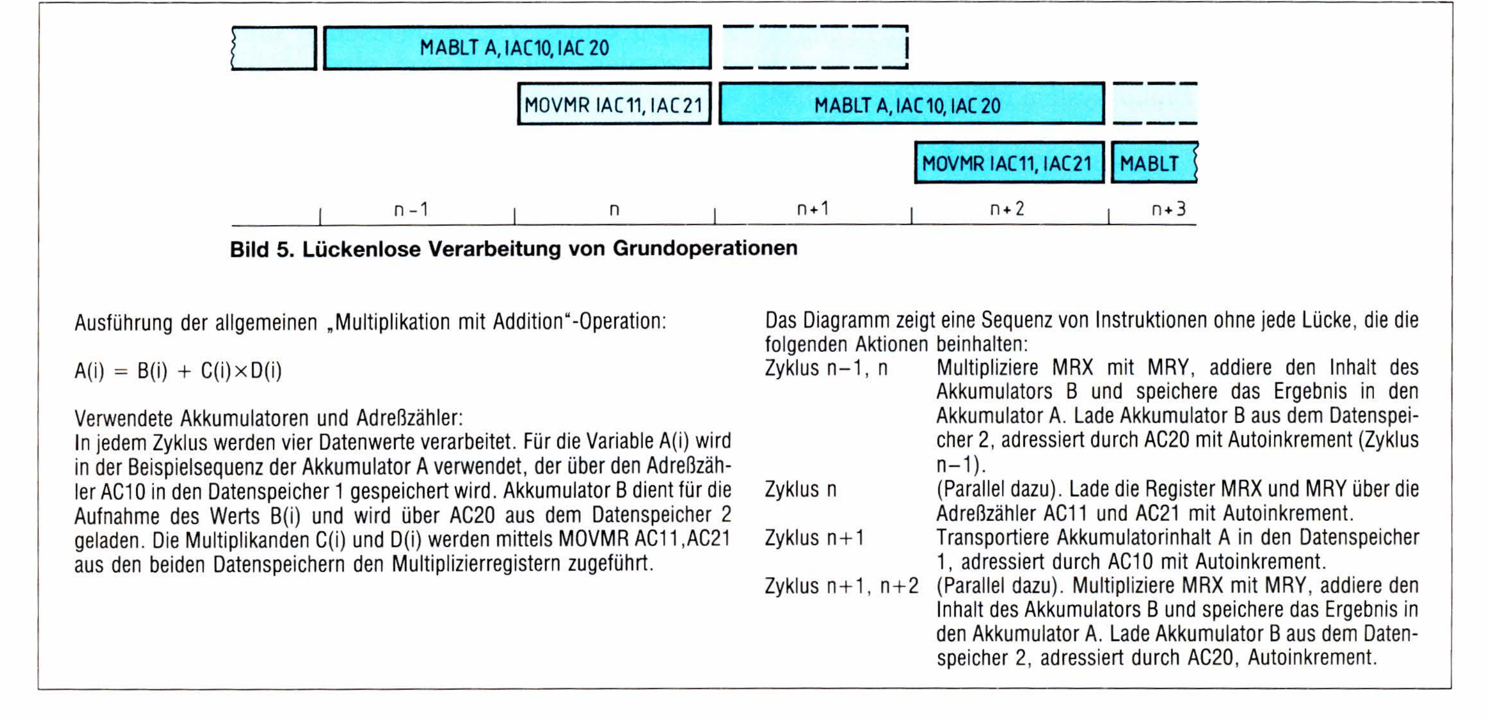


Bild 5. Lückenlose Verarbeitung von Grundoperationen

Ausführung der allgemeinen „Multiplikation mit Addition"-Operation:

A(i) = B(i) + C(i)×D(i)

Verwendete Akkumulatoren und Adreßzähler:
In jedem Zyklus werden vier Datenwerte verarbeitet. Für die Variable A(i) wird in der Beispielsequenz der Akkumulator A verwendet, der über den Adreßzähler AC10 in den Datenspeicher 1 gespeichert wird. Akkumulator B dient für die Aufnahme des Werts B(i) und wird über AC20 aus dem Datenspeicher 2 geladen. Die Multiplikanden C(i) und D(i) werden mittels MOVMR AC11,AC21 aus den beiden Datenspeichern den Multiplizierregistern zugeführt.

Das Diagramm zeigt eine Sequenz von Instruktionen ohne jede Lücke, die die folgenden Aktionen beinhalten:

Zyklus n−1, n	Multipliziere MRX mit MRY, addiere den Inhalt des Akkumulators B und speichere das Ergebnis in den Akkumulator A. Lade Akkumulator B aus dem Datenspeicher 2, adressiert durch AC20 mit Autoinkrement (Zyklus n−1).
Zyklus n	(Parallel dazu). Lade die Register MRX und MRY über die Adreßzähler AC11 und AC21 mit Autoinkrement.
Zyklus n+1	Transportiere Akkumulatorinhalt A in den Datenspeicher 1, adressiert durch AC10 mit Autoinkrement.
Zyklus n+1, n+2	(Parallel dazu). Multipliziere MRX mit MRY, addiere den Inhalt des Akkumulators B und speichere das Ergebnis in den Akkumulator A. Lade Akkumulator B aus dem Datenspeicher 2, adressiert durch AC20, Autoinkrement.

Tabelle 2. Beispielprogramm: Rekursives Filter zweiter Ordnung

```
;
;       REKURSIVER FILTERBLOCK 2TER ORDNUNG
;
RESET   JMP     INIT            ; INITIALISIERUNG, WIRD NACH
                                ;  'RESET' ANGESPRUNGEN
;
;
;       HAUPTSCHLEIFE: WIRD PRO ABTASTWERT EINMAL DURCHLAUFEN
;
SAMPLE  MOVMR   DAC10,IAC22     ; TRANSPORT X(N-2) UND B2
                                ;  ZUM MULTIPLIZIERER
        MUL     A               ; A = X(N-2) * B2
        MOVMR   DAC10,IAC22
        MAA     A               ; A = A + X(N-1) * B1
        MOVP    AC10,SDBF       ; X(N) EINLESEN, ABSPEICHERN
        MOVMR   DAC10,IAC22
        MAA     A               ; A = A + X(N) * B0
        MOVMR   DAC10,IAC22
        MAA     A               ; A = A + Y(N-2) * A2
        MOVMR   DAC10,AC22
        MAAT    A,DAC10         ; A = A + Y(N-1) * A1
                                ;  Y(N) ABSPEICHERN
        MOV     #AC22,TAB       ; POINTER FUER KOEFFIZIENTEN
                                ;  WIEDERHERSTELLEN
        NOP                     ; DARF BENUETZT WERDEN!
        MOVPH   SDBF,A          ; Y(N) AUSGEBEN
;
END     JMP     $               ; ENDLOSSCHLEIFE
;
;
;       INITIALISIERUNG: WIRD NACH JEDEM 'RESET' AUFGERUFEN
;
INIT    MOV     #AC10,4         ; INITIALISIEREN DER
        MOV     #AC11,4         ;  ADDRESSCOUNTER AC10, AC11
        MOV     #AC22,TAB       ;  (MODULO ADDRESSIERUNG)
                                ;  UND AC22 (KOEFFIZIENTEN)
        CLR     A
        MOV     #LC0,5
L0      MOVH    DAC10,A         ; SAMPLE-SPEICHER GLEICH
        DJNZ    LC0,L0          ;  NULL SETZEN
;
        JMP     $               ; ENDLOSSCHLEIFE
;
        END
```

über das serielle Interface. Sobald diese eintreffen, setzt der Prozessor die Programmausführung auf der Adresse 1 fort. Dort werden zunächst X(n–2) und X(n–1) verarbeitet, danach X(n) eingelesen, abgespeichert und verrechnet. Daran schließt sich die Bearbeitung der Werte Y(n–2) und Y(n–1) an. Das Ergebnis Y(n) wird abgespeichert und über das serielle Interface ausgegeben. Danach wartet der Prozessor wiederum in einer Endlosschleife auf das erneute Eintreffen von Daten, und die Aktion beginnt von neuem. Zu Anfang sind die alten Abtastwerte natürlich null, sie werden aber in den nächsten Zyklen durch eingelesene und berechnete Werte aufgefüllt.

Das Programm benützt die Adreßzähler AC10 und AC22. AC22 adressiert die Koeffizienten und wird nach jedem Durchgang wieder auf den Anfang der Tabelle gesetzt. Die Tabelle enthält der Reihe nach die Koeffizienten: B2, B1, B0, A2, A1. Der Adreßzähler AC10 wird zur Moduloadressierung verwendet: Er wird in einem Durchgang sechsmal dekrementiert, wobei er bei jedem Unterschreiten von null mit dem Inhalt des Zählers AC11 (damit mit vier) geladen wird. Da der Ringspeicher nur fünf Werte enthält, zeigt AC10 nach jedem Zyklus eine Adresse tiefer als zu Beginn.

Die Befehle MOVP und MOVPH benötigen zwei Instruktionszyklen, alle anderen nur einen. Damit beansprucht das Programm inklusive Ein- und Ausgabe 16 Instruktionszyklen, das sind 1,6 µs. Reine Rechenzeit für das Filter sind allerdings nur 10 Zyklen, d.h. 1 µs. Dies zeigt sich dann, wenn man sehr viele dieser Filter hintereinander schaltet, z.B. um einen Bandpaß hoher Güte herzustellen. Die Gesamtausführungsdauer berechnet sich nach der Formel: Anzahl der Filtersektionen × 1 µs + 0,4 µs für Ein-/Ausgabe und 0,2 µs für das Wiederaufsetzen des Koeffizientenpointers und ein NOP. Zehn solcher Filtersektionen (mit je fünf Multiplikationen) benötigen eine Rechenzeit von 10,6 µs inklusive Ein-/Ausgabe. Damit kann eine zehnfache Filterbank mit einer Abtastrate von 94 kHz arbeiten.

Bild 6 zeigt beispielhaft einen Frequenzgang eines digitalen Systems, das sich mit Hilfe eines UDPI 01 und dem obigen Programm realisieren läßt.

5.2 *Polynomapproximationen*

Auch andere Algorithmen lassen sich auf die Anwendung der Grundoperation umformen. Zur Verdeutlichung sei hier eine Methode gezeigt, mit der sich Polynome mit hoher Geschwindigkeit berechnen lassen. Gleichzeitig weist dieses Beispiel auf die Leistungsfähigkeit der Instruktionen MAALT und MABLT hin, mit deren Hilfe sich die allgemeine Grundoperation verwirklichen läßt. Gegeben sei das Polynom:

$$Y(x) = a_0 + a_1 \times x + a_2 \times x^2 + \ldots + a_n \times x^n$$

Durch Umformen nach Horners Schema erhält das Polynom eine zur Berechnung günstigere Struktur:

$$Y(x) = a_0 + x \times (a_1 + x \times (\ldots (a_{n-1} + a_n \times x) \ldots))$$

Die einzelnen Klammerausdrücke entsprechen damit genau der Grundoperation „Multipliziere und Addiere“. Die Berechnung eines Polynoms n-ter Ordnung erfordert n dieser Grundoperationen. Bei der tatsächlichen Realisation benötigt man zur Bearbeitung des nächsten Klammerausdrucks das Ergebnis des vorherigen. Dies zwingt normalerweise, auf das Multiplikationsergebnis zu warten. Man umgeht diese Unzulänglichkeit, indem

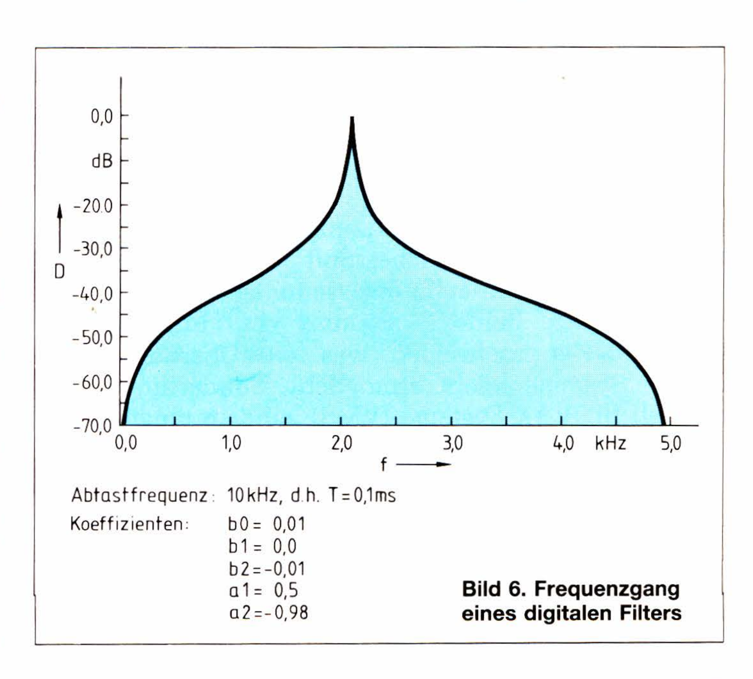


Bild 6. Frequenzgang eines digitalen Filters

man das Polynom nach dem Horner-Schema zweiter Ordnung in ein gerades und ein ungerades Polynom aufteilt:

$$Y(x) = a_0 + x^2 \times (a_2 + x^2 \times (\ldots (a_{n-2} + a_n \times x^2) \ldots)) + x \times (a_1 + x^2 \times (\ldots (a_{n-3} + a_{n-1} \times x^2) \ldots)) = P_g(x) + P_u(x)$$

Durch wechselseitige Berechnung der Klammerausdrücke in den Polynomen P_g und P_u stehen die benötigten Zwischenergebnisse immer rechtzeitig zur Verfügung. *Tabelle 3* zeigt einen Programmausschnitt mit den verwendeten Adreßzählern. Da obiges Schema n+1 Multiplikationen mit Additionen benötigt, können Polynome n-ter Ordnung in 2(n+1) Instruktionszyklen auf einem UDPI01 berechnet werden. Die Rechenzeit für ein Polynom 10-ten Grades beträgt z.B. nur 2,2 µs. Für Berechnungen nichtlinearer Funktionen (sin(x), exp(x), sqrt(x), usw.) ist der UDPI01 somit bestens gerüstet.

6 Leistungsmerkmale

Das wichtigste Kriterium zur Beurteilung eines digitalen Signalprozessors ist die Rechenleistung, wobei vorrangig eine Gegenüberstellung verschiedener DSPs interessant ist. Üblicherweise dient dazu die Angabe der Rechenzeit für bestimmte Aufgaben (z.B. rekursiver Block zweiten Grades, transversales Filter n-ter Ordnung, FET, usw.). Doch kann ein Vergleich nach diesen Kriterien nur oberflächlich bleiben, da zumeist wenig über die Rechengenauigkeit ausgesagt wird und die Angaben über die Aufgabe oft zu ungenau sind. Ein allgemeines rekursives Filter zweiter Ordnung umfaßt fünf Multiplikationen, oft wird jedoch nur von vier ausgegangen (d.h. ohne Skalierungsfaktor) oder gar nur von zwei Multiplikationen (reiner rekursiver Teil). Ohne eindeutige Angaben sind solche Vergleichstabellen wenig aussagekräftig.

Anhand der in *Tabelle 4* aufgeführten Rechenzeiten kann ein zukünftiger Anwender leicht den Rechenzeitbedarf eigener Programmentwicklungen abschätzen.

7 Entwicklungshilfsmittel

Die Entwicklungsunterstützung für den UDPI01 gliedert sich in zwei Bereiche: Soft- und Hardware. Die Software umfaßt einen Crossassembler und einen Softwaresimulator. Beide Programme sind in FORTRAN (Standard 77) geschrieben, was eine Übertragung auf andere Systeme leicht ermöglicht. Sonderfunktionen, speziell für VAX-Systeme (VMS), sind in einem Block zusammengefaßt, somit leicht adaptierbar. Der Crossassembler setzt die symbolische Assemblersprache des UDPI01 (siehe Programmbeispiele) in den ausführbaren Objektcode um. Der Simulator dient zum Austesten von Anwenderprogrammen, bevor sie in einem realen System tatsächlich zum Einsatz kommen. Da hierbei alle Aktionen des Prozessors simuliert werden, ist es z. B. möglich, Schritt für Schritt alle Aktionen während der Programmausführung zu überwachen. Der Software steht auf der Hardwareseite eine einfache Emulatorplatine gegenüber. Der Emulator basiert auf dem Emulatorchip UDPI EC, einer Version des UDPI 01 ohne Programm-ROM, dafür mit herausgeführtem Programmbus. Die auf Minirechner ausgetesteten Anwenderprogramme können entweder in PROMs geschrieben werden oder direkt in den Programmspeicher des Emulator-Boards. Damit lassen sich In-Circuit-Emulationen mit voller Geschwindigkeit ausführen.

8 Anwendungsbeispiele

Aufgrund der hohen Verarbeitungsrate bietet der UDPI 01 die Möglichkeit zur Ausführung einer Vielzahl von Anwendungen. Um einen Überblick darüber zu erhalten, sind an dieser Stelle einige Anwendungsbeispiele ohne Anspruch auf Vollständigkeit angeführt.

8.1 Sprachverarbeitung, Vocoder

Ein Vocoder dient zur Umsetzung von Sprachdaten auf eine niedere Datenrate. Das Sprachsignal wird nicht direkt codiert, sondern es werden bestimmte Modellparameter (Filterkoeffizienten, Grundfrequenz usw.) übertragen. Diese ändern sich wesentlich langsamer als die ursprüngliche Signalzeitfunktion. Sie können daher mit geringerer Frequenz abgetastet und über einen schmalbandigeren Kanal übermittelt werden (*Bild 7*). Diese

Tabelle 3. Berechnung eines Polynoms nach dem Horner-Schema 2ter Ordnung

Berechnet wird:

$Z_g = a_i + x^2 \times Z_g$
$Z_u = a_{i-1} + x^2 \times Z_u$

Verwendete Adreßzähler und die damit adressierten Daten:

```
AC10: a_i  ; Koeffizienten
AC11: x^2  ; Quadrat des Arguments
AC20: Z_g  ; Zwischenwert gerades Polynom
AC21: Z_u  ; Zwischenwert ungerades Polynom
```

Programmausschnitt:

```
MOVMR  AC11,AC20
MAALT  B,AC20,DAC10   ;Z_g = a_i + x^2 × Z_g
MOVMR  AC11,AC21
MAALT  B,AC21,DAC10   ;Z_u = a_i-1 + x^2 × Z_u
```

Tabelle 4. Ausführungszeiten von Testprogrammen

Testprogramm	Zeit
100faches transversales Filter (Akkumulation über 31 Bit, inklusive z^{-1}-Funktion und vorbereitende Aktionen)	20,1 µs
rekursives Filter 2ter Ordnung (vollständiges Filter: 5 Koeffizienten, Akkumulation über 31 Bit, Ergebnis 16 Bit, inklusive z^{-1}-Funktion und vorbereitende Aktionen)	1,2 µs
10 Filter-Sektionen rekursiver Filter 2ter Ordnung (siehe oben)	10,2 µs
Polynom 10ter Ordnung (11 Koeffizienten mit je 16 Bit)	2,2 µs
Sinusfunktion mit 14 Bit Genauigkeit	2,4 µs
komplexe FFT mit 64 Punkten	424,0 µs

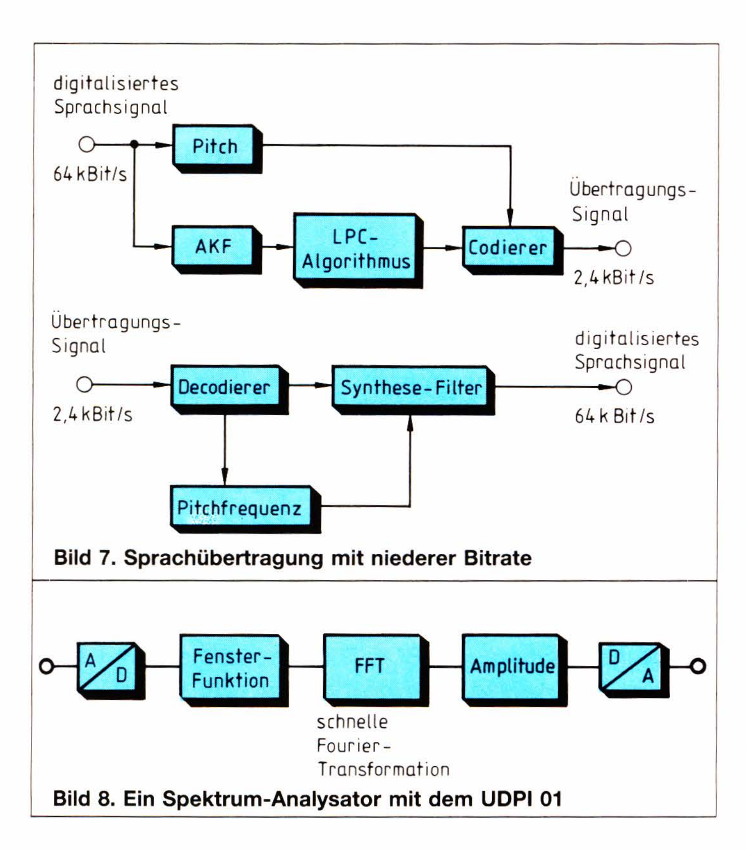


Bild 7. Sprachübertragung mit niederer Bitrate

Bild 8. Ein Spektrum-Analysator mit dem UDPI 01

Codierung kann eine Datenverringerung bis zu einem Faktor 30 liefern. Auf der Empfängerseite dienen die übermittelten Parameter zur Steuerung der Sprachsynthese. Mit dem UDPI01 kann ein Vocoder-System aufgebaut werden, das außer einem UDPI01 selbst noch einen Steuerprozessor (z.B. 68000), einen Speicher und E/A-Bausteine benötigt. Die Eigenschaften dieses LPC-Vocoders sind:

- Voll-Duplex-Vocoder
- Rahmenlänge 200 Abtastwerte (25 ms)
- Abtastrate 8 kHz
- 10 LPC-Koeffizienten
- 140 Werte der Autokorrelationsfunktion.

8.2 *Spektrum-Analysator*

Ein System mit je einem D/A- bzw. A/D-Umsetzer und einem UDPI01 ist in der Lage, eine komplette Spektralanalyse durchzuführen. Die Aktionen des Signalprozessors beinhalten dabei das „Windowing“, um Leckeffekte bei der FFT zu mindern, die eigentliche 64-Punkte-FFT und eine nachfolgende Verarbeitung, welche die Amplitudenwerte extrahiert (*Bild 8*).

8.3 *Modem*

Der UDPI 01 kann als 1,2-kBit-FSK-Modem programmiert werden (*Bild 9*). Es umfaßt dann per Programm den FSK-Modulator, das Sendefilter, das Empfangsfilter, den Amplitudendetektor, den Frequenzdetektor und den Demodulator.

8.4 *Echounterdrückung*

Eine Echounterdrückung ist ein typischer Anwendungsfall eines adaptiven Filters. Hierzu ein Beispiel aus der Telekommunikation. Auch dort setzen sich immer mehr digitale Systeme durch. In der Zukunft wird die Digitalisierung über die Teilnehmerleitung bis zum Teilnehmer geführt werden. Für die nahe Zukunft wird für die Digitalisierung des Telefonnetzes davon ausgegangen, daß die vorhandenen Kabel weiter verwendet werden. Damit auf der 2-Draht-Teilnehmerleitung eine fehlerfreie Übertragung in beiden Richtungen möglich ist, muß das von einem Teilnehmer zur Vermittlungsstelle laufende und dort teilweise reflektierte Signal (Echo) so kompensiert werden, daß im Empfangskanal nur noch das Sendesignal des anderen Teilnehmers übrigbleibt. Da die Übertragungseigenschaften der einzelnen Teilnehmerleitungen unterschiedlich sind und außerdem noch zeitlichen Schwankungen unterliegen, muß das Kompensationsverfahren adaptiv sein, d.h. die einzelnen Filterkoeffizienten des Echokompensators müssen sich an die Leitungseigenschaft anpassen.

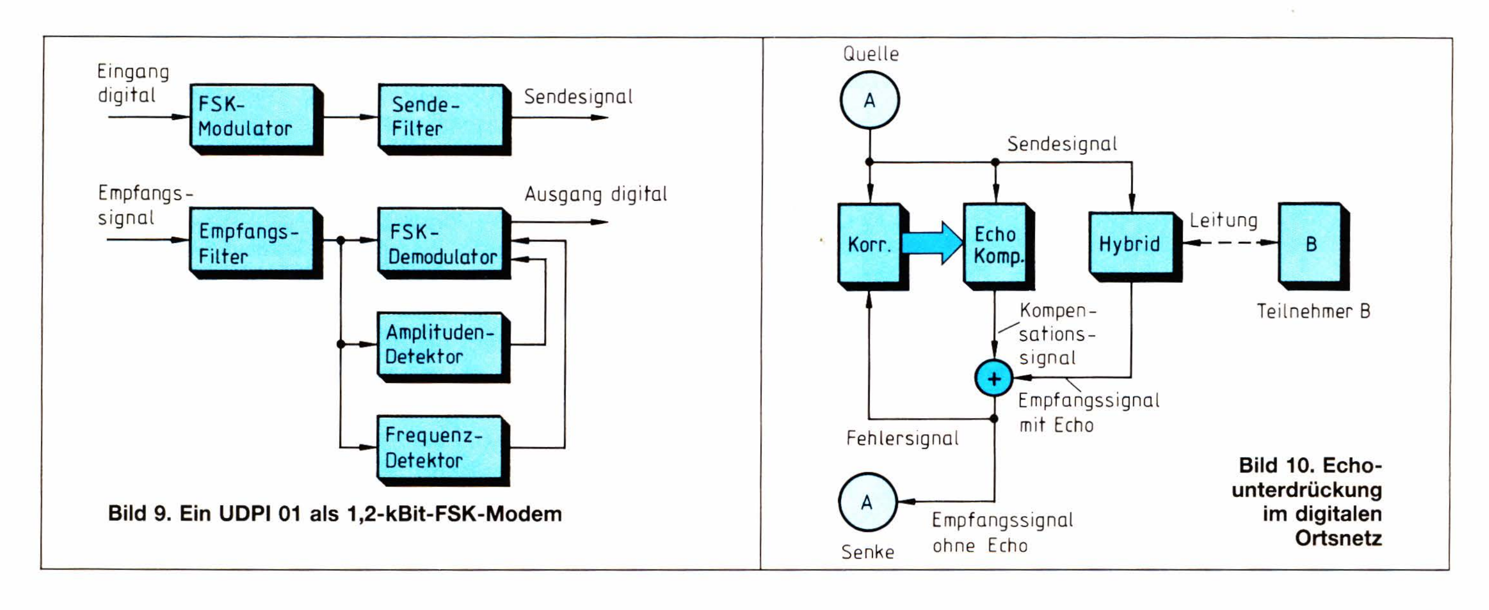


Bild 9. Ein UDPI 01 als 1,2-kBit-FSK-Modem

Bild 10. Echounterdrückung im digitalen Ortsnetz

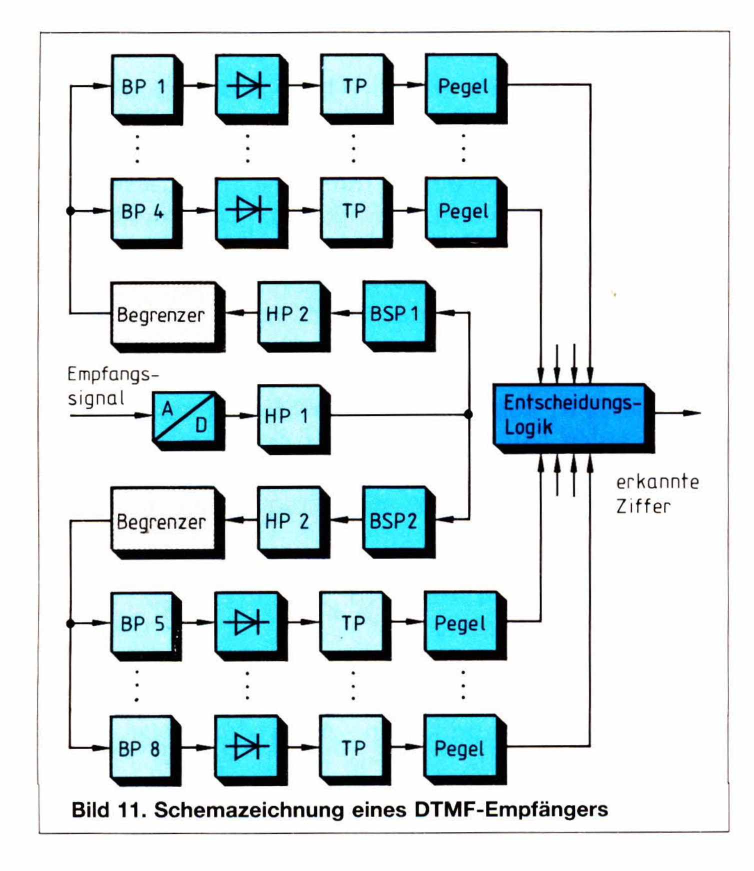


Bild 11. Schemazeichnung eines DTMF-Empfängers

Diese Aufgabe kann ein als Echounterdrücker programmierter UDPI 01 übernehmen (*Bild 10*).

8.5 *DTMF-Empfänger*

Ein UDPI 01 kann verwendet werden, um aus dem digitalisierten Sendesignal eines Telefonteilnehmers die entsprechenden Frequenzanteile herauszufiltern und daraus den Tastencode zu ermitteln (*Bild 11*). Die gesamte Verarbeitung umfaßt mehrere Hoch-, Tief- und Bandpaßfilter, Bandsperren, Begrenzer, Gleichrichter und Diskriminatoren sowie eine Entscheidungslogik. Mehrfach benötigte Funktionsblöcke werden dabei im Zeitmultiplex durchlaufen (Abtastrate: 10 kHz).

8.6 *Audio-Verarbeitung*

Interne Wortlängen und die Verarbeitungsgeschwindigkeit des UDPI 01 sind ausreichend konzipiert, um eine komplette Tonverarbeitung in TV-Geräten oder auch in HiFi-Radio-Systemen zu übernehmen. Mit entsprechendem Programm könnte ein UDPI 01 z.B. folgende Aufgaben übernehmen (*Bild 12*):

- Dematrizierung
- sin(x)/x-Entzerrung
- Lautstärke
- Höhen- und Tiefenregelung
- Physiologie
- Balance
- Stereo-Basisbreite
- Pseudo-Stereo
- Stereo/Mono-Umblendung.

  Zusätzlich kann eine Reihe von Nichtaudiofunktionen softwaremäßig realisiert werden:
- Stereoerkennung
- Erkennung Stereo/zweisprachig/Mono für TV-Geräte
- Verkehrsfunkdecoder für Autoradio
- Stereo-PLL usw.

Literatur

[1] *Azizi, S.A.*: Entwurf und Realisierung Digitaler Filter, Oldenbourg Verlag.
[2] *Lacroix, A.*: Digitale Filter, eine Einführung in zeitdiskrete Signale und Systeme, Oldenbourg Verlag.
[3] *Rabiner, L.R.; Gold, B.*: Theory and Application of Digital Signal Processing, Prentice-Hall, Inc. Englewood Cliffs, New Jersey (1975).
[4] *Pfeifer, H.*: Digitale Signalverarbeitung, Elektronik Industrie 5 (1984), S. 14...18.

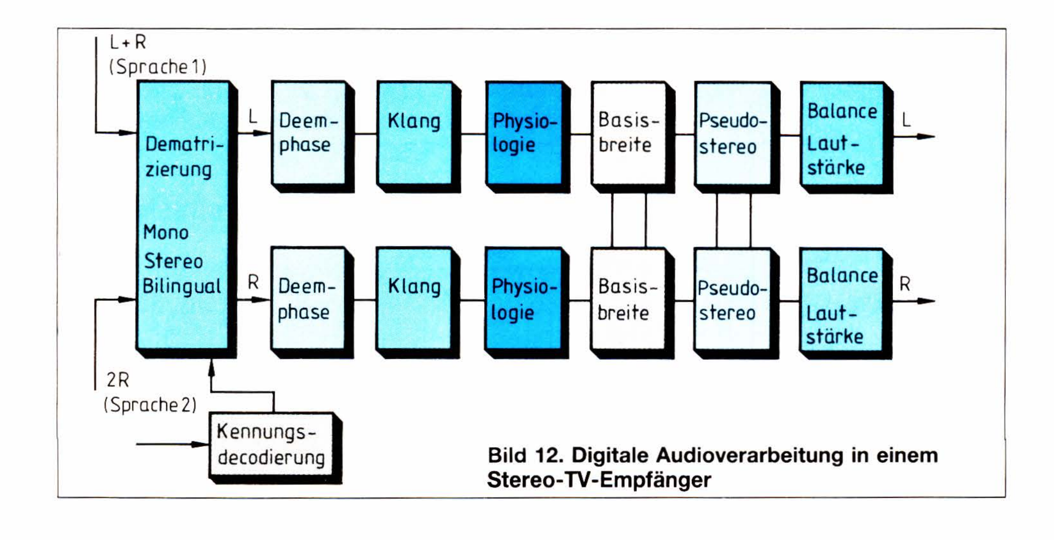
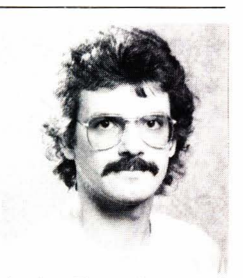


Bild 12. Digitale Audioverarbeitung in einem Stereo-TV-Empfänger

Dipl.-Phys. Martin Winterer stammt aus Esslingen a. N. und ist aufgewachsen in Hausach i. K. Nach seinem Physikstudium an der Universität Freiburg übernahm er eine Arbeitsstelle bei der Firma Intermetall GmbH, Freiburg, wo er seit November 1983 in der Abteilung „Concept Engineering" tätig ist. Seine Arbeitsgebiete umfassen allgemeine Probleme der digitalen Signalverarbeitung mit den Schwerpunkten Telekommunikation und Audioverarbeitung.
Hobbxs: Windsurfen, Wandern, Skifahren und Radfahren.



Wilfried Taetow

1. Teil

CMOS-Bausteine für mikroprogrammierbare Signalprozessoren

Die CMOS-Technologie erobert eine weitere bipolare Domäne – die schnelle digitale Signalverarbeitung. Eine Familie aus sechs CMOS-Bausteinen ermöglicht den Aufbau von mikroprogrammierbaren Signalprozessoren, die im Hinblick auf Geschwindigkeit und Leistungsfähigkeit bisherige bipolare Systeme weit übertreffen. Durch jüngste Fortschritte in der Halbleitertechnologie gerät die digitale Signalverarbeitung mehr und mehr ins Blickfeld der Entwicklungsingenieure. Systeme zur digitalen Verarbeitung von Ton-, Bild- oder Funksignalen, die bislang ausschließlich theoretisch entworfen werden konnten, sind durch die Bereitstellung von immer komplexeren Bausteinen in den Bereich des Möglichen gerückt.

Status quo

Ein typisches digitales Signalverarbeitungssystem führt einen Algorithmus mit Zahlen durch, die aus analogen Abtastwerten stammen. Der Algorithmus wiederholt viele Male Instruktionen, die komplizierte Berechnungen mit diesen Daten durchführen. Mit der Leistungsfähigkeit der Algorithmen wächst natürlich auch die Anzahl der Berechnungen, oft sogar überproportional. Dies erfordert eine Durchsatzrate (Operationen pro Zeiteinheit), die mit Standard-Mikroprozessoren nicht zu erreichen ist. In der Tat müssen alle Funktionsblöcke eines Signalprozessors wie Programm-Sequenzer, Adreß-Generator und arithmetische Elemente die gleiche hohe Arbeitsgeschwindigkeit aufweisen.

Den grundsätzlichen Aufbau eines Signalprozessors zeigt *Bild 1*. Der Programm-Sequenzer steuert den Befehlsfluß innerhalb des Prozessors. Er bestimmt, welche Adresse als nächstes an den Programmspeicher gegeben wird. Die Speicherinhalte steuern dann die übrigen Bausteine des Systems. Außerdem ist der Sequenzer für Verzweigungen, Unterprogrammsprünge und Interrupts verantwortlich. Außer auf Schnelligkeit sollte der Sequenzer auch daraufhin entworfen sein, Schleifen ohne Leerzeiten abzuarbeiten und natürlich auf keinen Fall das Gesamtsystem verlangsamen. Der Adreß-Generator spezifiziert die Speicherplätze im Datenspeicher, der alle Ein- und Ausgabedaten sowie Konstanten (Koeffizienten) enthält. Da viele Algorithmen, die in der digitalen Signalverarbeitung verwendet werden, eine komplizierte Adreß-Struktur haben sowie einen schnellen Datentransfer zu und von den Arithmetik-Bausteinen benötigen, ist ein flexibler Adreß-Generator hoher Geschwindigkeit unbedingt notwendig.

Die eigentliche Rechentätigkeit wird von arithmetisch logischen Einheiten (ALU), Multiplizierern und Barrel-Shiftern geleistet. Deren Durchsatzrate ist eine Funktion von Geschwindigkeit interner Architektur und Ein-/

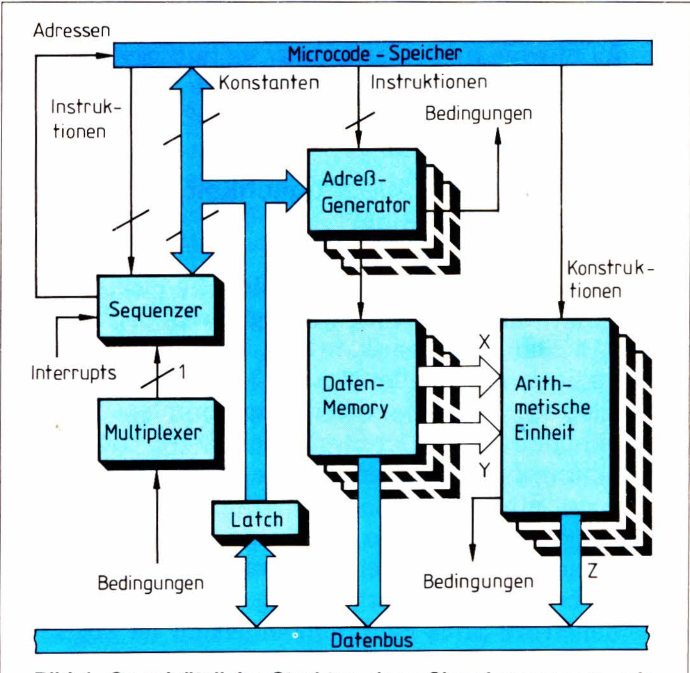


Bild 1. Grundsätzliche Struktur eines Signalprozessors, wie sie auch mit Bausteinen der Word-Slice-Familie realisiert werden kann

Ausgabestruktur. Selbstverständlich müssen sie die binäre Arithmetik schnell, genau und effizient beherrschen.

Digitale Signalprozessoren, die alle diese Elemente auf einem einzigen Chip vereinigen, sind für preissensitive Anwendungen im unteren Leistungsbereich gedacht. Uneffiziente Schleifenverarbeitung, Ein-/Ausgabeengpässe, eingeengte Instruktionsworte und unflexible Architektur beeinträchtigen erheblich die Durchsatzrate. Bipolare Bit-Slice-Architekturen besitzen die notwendige Geschwindigkeit und Flexibilität für eine hohe Leistungsfähigkeit. Sie leiden aber unter der hohen Anzahl der notwendigen Bausteine, verbrauchen eine enorme Leistung und beanspruchen nicht zuletzt einen erheblichen Platz. Zudem vergrößert die notwendige externe Beschaltung mit SSI- und MSI-Bausteinen die Programmierungskomplexität und kann zu wahren „Verstrickungen" im Signalfluß führen.

Die spezifischen Eigenschaften solcher Systeme lassen sich auf die individuellen Bausteine zurückverfolgen. So können z.B. die gängigen Programm-Sequenzer keinen Interrupt auf dem Chip behandeln oder gar speichern. Sie können nur einen recht begrenzten Programmspeicher bedienen und haben eine bescheidene Stack-Tiefe. Was aber am schlimmsten ist, sie sind in ihrer Geschwindigkeit nicht den fortschrittlichen Arithmetikbausteinen angepaßt und begrenzen damit die Systemgeschwindigkeit. Komplette Adreß-Generatoren gibt es bis jetzt nur für spezielle Algorithmen (FFT).

Universelle Adreß-Generatoren z.B. für 16 Bit müssen aus vier ALU-Bausteinen mühsam aufgebaut werden. Momentan erhältliche Multiplizierer und ALUs benötigen die Unterstützung von externen Schaltkreisen. Multiplizierer brauchen sie für offsetfreie Rundung, Formatierung, Rückkopplung und Speicherung. Multiport-ALU-Bausteine verbrauchen einige Watt, und ihre Funktionalität ist ohne eine beträchtliche Anzahl von SSI-Schaltkreisen nur sehr begrenzt.

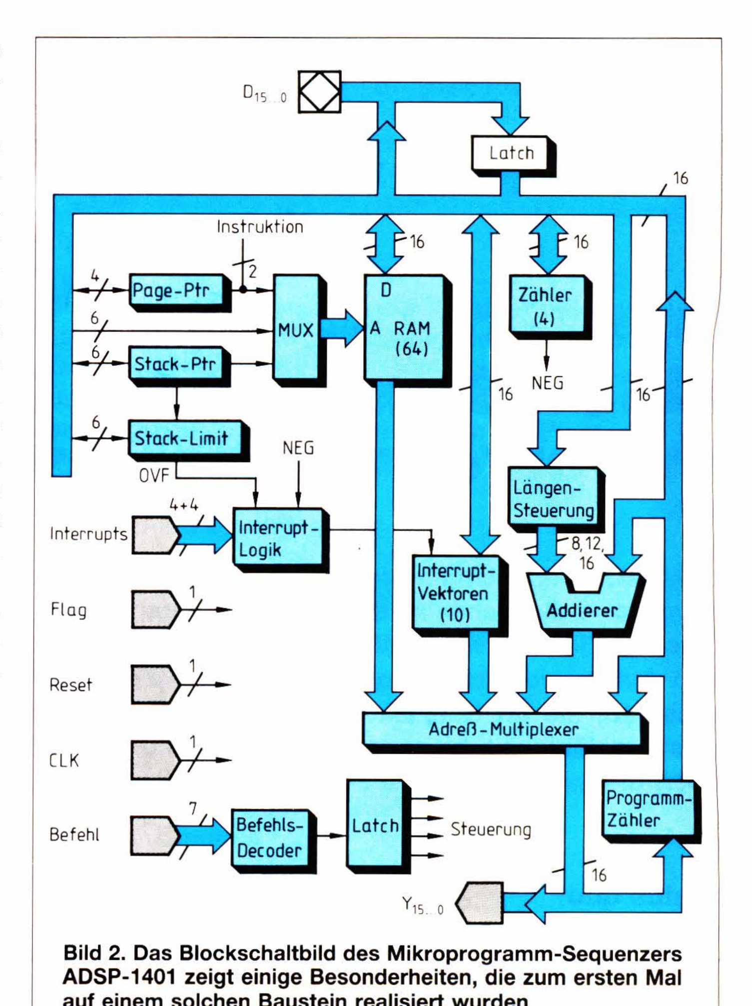


Bild 2. Das Blockschaltbild des Mikroprogramm-Sequenzers ADSP-1401 zeigt einige Besonderheiten, die zum ersten Mal auf einem solchen Baustein realisiert wurden

Neue Bausteine – neue Möglichkeiten
Die „Word-Slice-Familie"

Seit ca. zwei Jahren beschäftigt sich die Firma Analog Devices mit Bausteinen für die digitale Signalverarbeitung. Mit Standardprodukten wie Multiplizierern und Multiplizierer-Akkumulatoren in CMOS-Technik wurde der technologische Grundstein für komplexere Funktionseinheiten gelegt. Der zweite Schritt war die Weiterentwicklung der Halbleitertechnologie von einem 5-µm-Prozeß mit einer Metallschicht zu einem 2-µm-Prozeß mit zwei Metallschichten, was eine Verdopplung der Geschwindigkeit und Verkleinerung der Chip-Fläche mit sich brachte. Jetzt war der Weg frei für die Entwicklung der Word-Slice-Familie. Der Name ist abgeleitet von Bit-Slice-Bausteinen, wie sie seit Jahren zum Aufbau von mikroprogrammierbaren Signalprozessoren verwendet werden. Die beiden wesentlichsten Unterschiede sind jedoch CMOS- statt bipolarer Technologie und Wortbreiten von 16 bzw. 32 Bit statt 4 Bit pro Baustein. Die Familie besteht aus einem Programm-Sequenzer, der einen sehr großen Adreß-Raum ansteuern kann, einem Adreß-Generator, der universell verwendbar ist, einer arithmetisch logischen- und Schiebeeinheit (ALSU), die einen Barrel-Shifter parallel zur ALU enthält und einen Multiplizierer-Akkumulator, der um einige zusätzliche Funktionen erweitert ist. Zwei weitere Bausteine sind für die Verarbeitung von Gleitkommazahlen vorgesehen. Ein ALU-Baustein und ein Multiplizierer können sowohl einfache als auch doppeltgenaue Operationen durchführen. Alle sechs Bausteine sind von ihrer Struktur und ihrem Timing her so ausgelegt, daß ein Durchsatz von 10 Mio. Operationen pro Sekunde (10 MOPS bzw. 10 MFLOPS) auf jeden Fall gewährleistet ist. Neben ihrer hohen Durchsatzrate und großen Wortbreite bietet die Word-Slice-Familie die Flexibilität zum Aufbau von universellen digitalen Signalverarbeitungssystemen. Mit anderen Worten: Diese Bausteine können zu Prozessoren konfiguriert werden, die im Hinblick auf Durchsatz optimiert sind, und zwar nicht nur für eine spezielle Anwendung, sondern für ein breites Spektrum von Algorithmen. (Im 2. Teil dieses Beitrags werden verschiedene Architekturen auf ihre Leistungsfähigkeit untersucht.) Ohne daß Kompromisse

bei der Flexibilität eingegangen wurden, sind bereits viele Funktionen auf den Chips mit integriert, die früher durch externe Hardware realisiert werden mußten. Die konsequente Reduzierung dieser verbindenden Logik beschleunigt den Systementwurf und spart außerdem Platz und Leistung.

Alles unter Kontrolle

Das erste Mitglied der Familie ist der ADSP-1401, ein 16-Bit-Mikroprogramm-Sequenzer (*Bild 2*) mit einer Zykluszeit von 90 ns. Dieser 48polige Baustein spricht einen größeren Adreßraum an als alle anderen verfügbaren Sequenzer und ist der einzige, der in der Lage ist, Interrupts vollständig auf dem Chip zu speichern und zu verarbeiten. Der Sequenzer bestimmt die aufeinanderfolgenden Adressen auf der Basis von Instruktionen, Bedingungen und Interrupts, die ihm zur Verfügung gestellt werden. Jede Adresse, die er erzeugt, kann die momentane Adresse plus 1, eine absolute oder relative Adresse, die aus einer externen Quelle stammt, einer von 10 maskierbaren und priorisierten Interrupt-Vektoren, eine Sprungadresse aus dem internen RAM oder eine Adresse vom Top-of-Stack sein. Der Befehlssatz des Sequenzers ist in einem 7-Bit-Mikrocodewort enthalten, das 12 bedingte Befehle einschließlich Ein-, Zwei- und Dreiweg-Verzweigungen, Unterprogramm-Verzweigungen und Rückkehrbefehle enthält. Diese 12 Instruktionen können auf eine von vier Bedingungen ansprechen: Ein unbedingter Sprung; ein Sprung, wenn einer der vier internen Zähler unter Null herabzählt; oder die An- bzw. Abwesenheit eines Flags von einem externen Condition-Code-Selector. Trotz dieser eindrucksvollen Verzweigungsmöglichkeiten ist die interessanteste Eigenschaft des Sequenzers sein 64×16-Bit-RAM. Dieser Speicherbereich kann einer Vielzahl von Anwendungen dynamisch zugeordnet sein, einschließlich Stack, Sprungadressenspeicher und Zählerzwischenspeicher. Dem RAM kann komplett eine Funktion zugewiesen (z. B. 64 Worte, tiefer Stack) oder es kann auf zwei Funktionen aufgeteilt werden (z.B. 20 Worte Stack, 44 Sprungadressen). Der Stack-Bereich beginnt bei der Speicherzelle 0 und reicht bis zu der Zahl, auf die der Stack-Limit-Pointer zeigt. Push-Befehle, die den Stack-Pointer über das spezifizierte Limit hinausschieben, erzeugen automatisch einen Interrupt.

Bild 3. Universell verwendbar für alle Algorithmen der digitalen Signalverarbeitung ist der Adreß-Generator ADSP-1410

Die Fähigkeit des Sequenzers, Interrupts auf dem Chip zu speichern und zu verarbeiten, spart sowohl Platz als auch Leistung und umgeht die inhärenten Verzögerungen externer Schaltkreise. Alle zehn Interrupts sind maskierbar sowie priorisiert und werden in dem ihrem Auftreten folgenden Taktzyklus bearbeitet. Acht bedienen externe Quellen, die anderen zwei stammen aus dem Chip selbst (Stack-Überlauf und Zähler unter Null).

Vier abwärtszählende 15-Bit-Zähler dienen zur Verfolgung von Schleifen und Ereignissen. Sollte ein Programm fünf oder mehr Zähler benötigen, so kann der Zählerstand vorübergehend im RAM ausgelagert werden. Anders als viele andere Sequenzer kann der ADSP-1401 zum Laden von Befehlen aus externen Programmspeichern in schreibfähige Mikroprogrammspeicher („Writable Control Store“) benutzt werden. Solche Speicher sparen Leistung und Kosten, da sie es dem Sequenzer erlauben, seine Befehle direkt aus einem schnellen RAM zu beziehen, während der Großteil des Programms in langsamen und billigen ROMs gespeichert ist. Ebenso einzigartig ist die Möglichkeit, die Inhalte aller internen Register von außen zu lesen und in sie hineinzuschreiben, was besonders bei der Programmentwicklung eine wichtige Eigenschaft ist.

Komplett, aber universell

Das zweite Mitglied der Word-Slice-Familie ist der Typ ADSP-1410, ein schneller universeller Adreß-Generator. Mit seiner 16-Bit-Dual-Port-Architektur, seiner Zykluszeit von 70 ns und seiner internen Registerstruktur ist dieser 48polige CMOS-Baustein der erste, der all die „kniffligen" Adressierungen in der digitalen Signalverarbeitung bewältigt (*Bild 3*). So erzeugt er leicht alle Adressen für den Datenspeicher, die bei Routinen, wie der schnellen Fourier-Transformation, digitalen Filterung und Matrizenoperationen, benötigt werden. In einem einzigen Befehlszyklus bewältigt er vier fundamentale Operationen. Er kann eine Adresse zum Datenspeicher schicken, sie intern mit einem Offset-Wert modifizieren, um die nächste Schreib- oder Leseadresse zu berechnen, den ausgegebenen Wert mit einem vorgegebenen vergleichen, und Programmschleifen, basierend auf diesem Vergleich, reinitialisieren. Erfordert eine Anwendung einen größeren Adreß-Bereich als 64 K Worte, so kann der Anwender zwei Adreß-Generatoren ohne Geschwindigkeitseinbuße kaskadieren (1 G Worte). Dies wird dadurch ermöglicht, daß das MSB des Y- und D-Ports speziell dafür vorgesehen ist, Übertrags- oder Gleichheits-Flags zwischen beiden Chips zu übertragen. Alternativ kann ein einzelner Adreß-Generator dieselbe breite Adressierung leisten, indem er zwei Zyklen pro Adresse verwendet. Der ADSP-1410 enthält 30 Register mit jeweils 16 Bit, die in vier Gruppen aufgeteilt sind. 16 Adreß-Register (R), vier Initialisierungsregister (I), sechs Offset-Register (B) und vier Vergleichsregister (C). Das 10 Bit breite Instruktionswort enthält Befehle für Schleifen, Adreß-Modifikation, Registerlesen oder -schreiben und interne Transfers. So wird z.B. folgende Befehlszeile in einem einzigen Zyklus abgearbeitet.

$Y := R_n$; IF $R_n = C_j$ THEN $R_n := I_j$; ELSE $R_n := R_n + B_m$

Alle Befehle kommen normalerweise aus externen Quellen (Mikroprogrammspeicher). Der Adreß-Generator verfügt jedoch über ein „Alternate Instruction Register" (AIR), dessen Inhalt die externe Instruktion ersetzt, sobald es über „AIR ENABLE" aktiviert wird. Der Sinn dieses Registers ist es, Mikrocode einzusparen. So kann z.B. eine häufig verwendete Instruktion ins AIR geladen und später durch ein einziges Bit aufgerufen werden. Auch wenn mehrere Adreß-Generatoren von demselben Mikrocode angesteuert werden, kann es notwendig sein, einen einzelnen Generator vorübergehend einen abweichenden Befehl ausführen zu lassen.

Daß auch die speziellen Bedürfnisse einer FFT berücksichtigt wurden, zeigt der „Bit-Reverse-Multiplexer".

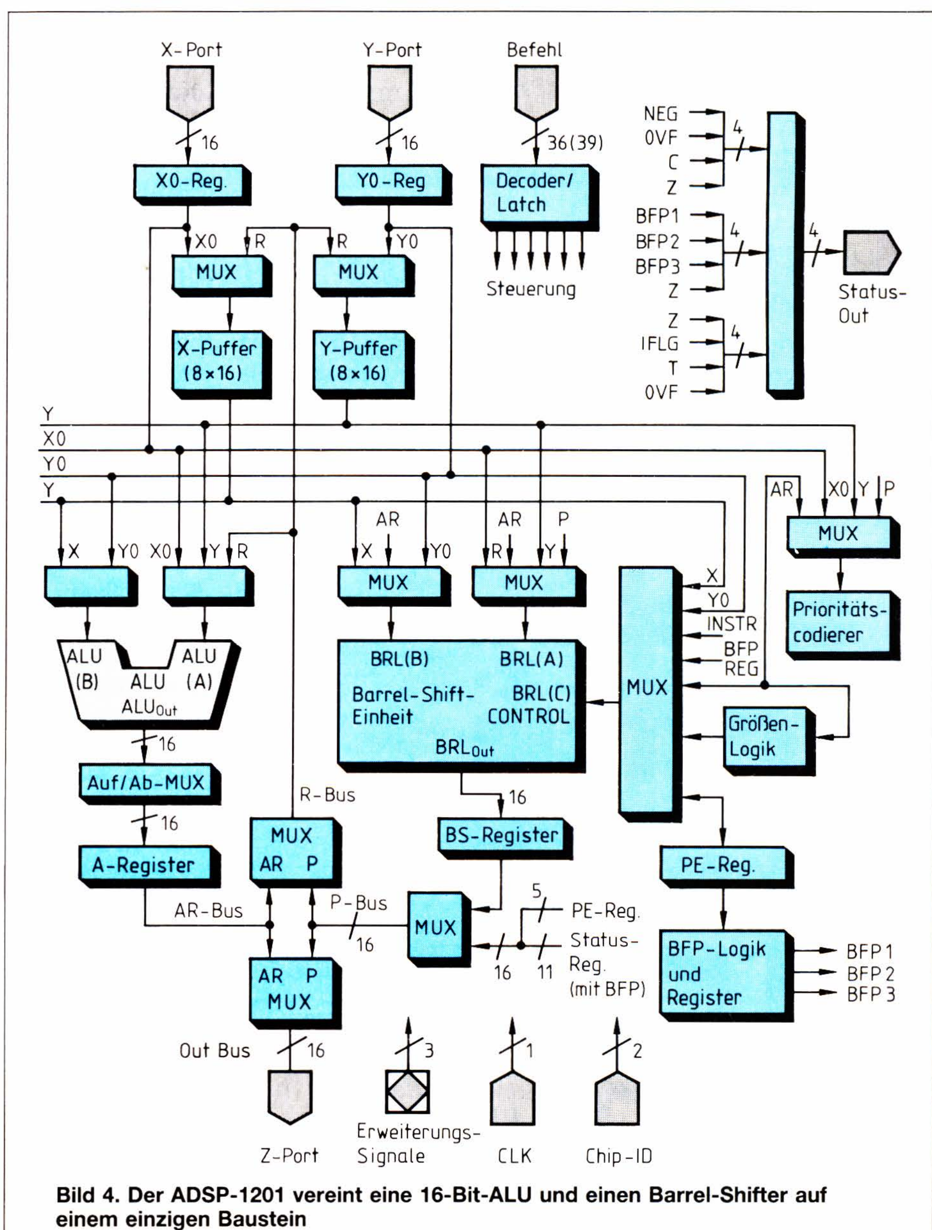
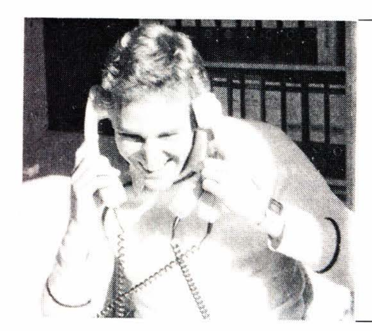


Bild 4. Der ADSP-1201 vereint eine 16-Bit-ALU und einen Barrel-Shifter auf einem einzigen Baustein

Addieren und Schieben

Das dritte Familienmitglied, der ADSP-1201, ist eine arithmetisch-logische Schiebeeinheit (ALSU). Mit diesem Chip wird eine Lücke geschlossen, die Entwicklern von Hochleistungssystemen viel Kopfzerbrechen berei-



Dipl.-Ing. Wilfried Taetow ist in Bensheim an der sonnigen Bergstraße geboren. Er studierte an der Technischen Hochschule in Darmstadt Nachrichtentechnik und verteidigte danach 15 Monate sein Vaterland. Seit Juni 1981 ist er bei der Firma Analog Devices in München als Vertriebsingenieur tätig. Seine Hobbys sind sein Mikrocomputer, Sport in Maßen (Radeln, Squash) und Satire (siehe Foto).

tet hat. In einem einzigen Taktzyklus von 60 ns kann sie zwei 16-Bit-Operanden verarbeiten und das Ergebnis ausgeben. Dieser Baustein ist nicht nur schnell, sondern außerdem mit einem unabhängig steuerbaren 32-Bit-Barrel-Shifter, zwei je 8 Worte tiefen Registern, zwei Eingangsregistern und einem Prioritäts-Codierer versehen (*Bild 4*). Das 36 Bit breite Befehlswort gibt dem Entwickler völlige Kontrolle über die interne Hardware. Das Instruktionswort ist jedoch in feste, wohldefinierte Felder aufgeteilt, so daß der Anwender je nach Aufgabe eine mehr oder weniger große Anzahl von Bits fest verdrahten kann und somit Mikrocode-Wortbreite spart. Die Hauptelemente des Chips sind die arithmetisch-logische Einheit und der Barrel-Shifter. Jedes dieser beiden Elemente arbeitet völlig unabhängig voneinander und zur gleichen Zeit. Jedes kann direkt mit den Eingangsdaten arbeiten oder das Ergebnis der ALU kann als Eingangswert für den Barrel-Shifter dienen und umgekehrt. Ebenso kann sich der Ausgangswert des Chips aus dem Ergebnis der einen als auch der anderen Einheit ergeben. Anders als übliche Bausteine bietet der ADSP-1201 die Möglichkeit, Daten mit gleicher Effizienz vor und nach einer arithmetisch-logischen Operation zu verschieben.

Selbstverständlich beherrscht die ALU alle arithmetischen und logischen Standardfunktionen. Das Ergebnis der Operation kann jedoch im selben Zyklus noch um 1 Bit nach links oder rechts verschoben werden. Weiterhin können die meisten Operationen in Abhängigkeit von einem internen Flag (IFLG) durchgeführt werden. Dieses Flag wiederum kann von einem der folgenden Flags gesetzt werden:
Überlauf (OVF), negativ (NEG), Null (Z), Vorzeichen (SGN), Bit-Test oder einer der Block-Floating-Point-Flags.

Zwei Arten von bedingten Operationen sind möglich. Im ersten Fall führt der Chip entweder einen Befehl oder keinen Befehl (NOP) aus, im zweiten Fall kann er aber auch statt NOP die inverse Instruktion ausführen, z. B.:

IF Flag THEN A + B ELSE NOP oder
IF Flag THEN A + B ELSE A – B

Diese zweite Möglichkeit der bedingten Befehle kann viele Zyklen in arithmetischen Berechnungen einsparen. Zum Beispiel, wie im 2. Teil besprochen wird, spielen Additionen mit darauffolgenden Schiebeoperationen in Abhängigkeit von Block-Floating-Point-Flags bei der FFT eine entscheidende Rolle. Eine effiziente Block-Floating-Point-Arithmetik kann den Signal/Rauschabstand um einige dB verbessern und Überlaufprobleme vermeiden.

Die Kombination von arithmetischen Operationen und bedingter Verschiebung beschleunigt ebenfalls Divisions-Algorithmen, die jeweils in 16 Zyklen durchgeführt werden können. Seine überragende arithmetische Leistungsfähigkeit erlangt dieser Chip jedoch erst durch den bereits erwähnten Barrel-Shifter. Zur Anschauung stelle man sich ein 48 Bit breites Feld vor, das in drei Segmente à 16 Bit aufgeteilt ist. Der Operand B sei in Segment 1 und 3 enthalten, der Operand A im mittleren Segment 2. Über das gesamte Feld kann nun eine 16 Bit breite Maske in einem Schritt beliebig nach rechts oder links verschoben werden. Eine geschickte Wahl sowohl der Quellregister als auch der Shift-Steuerung ermöglicht eine Vielzahl von Operationen einschließlich arithmetischer und logischer Verschiebungen um n Bit, automatischer Normalisierung von Eingangswerten, Rotationen, Block-Floating-Point-Arithmetik und Skalierung.

Zwei unabhängige, je 8 Wort tiefe und getrennt steuerbare Registergruppen unterstützen ALU und Barrel-Shifter. Jedes Register kann separat gelesen oder geschrieben werden. Da außerdem jede Operation, die mit dem A-Eingang der ALU durchgeführt werden kann, ebenso mit Eingang B möglich ist, bedingt diese Registerstruktur keinerlei programmtechnische Restriktionen. Anders als bei Bausteinen mit nur einer einzigen Registergruppe ist es hier sehr wohl möglich, zwei Operanden aus dem Chip-Speicher in einem Zyklus zu verarbeiten.

Da jeder numerische Prozessor sowohl mit vorzeichenbehafteten als auch mit vorzeichenlosen Zahlen rechnen soll, sind alle Funktionseinheiten des Chips in der Lage, sowohl mit 2-Komplement als auch mit reinen Betragszahlen zu arbeiten. So ermittelt z. B. der Prioritäts-Codierer das signifikanteste Bit eines Wortes in den beiden Formaten. Dadurch kann dessen Ausgang den Barrel-Shifter steuern und Zahlen beider Formate automatisch in einem Zyklus normieren. Außerdem kann er die betragsmäßig größte Zahl in einem Datenfeld ermitteln (siehe Teil 2). Mit wenigen Ausnahmen besteht auch die Möglichkeit, alle ALU- und Barrel-Shift-Operationen so zu erweitern, daß mit zwei Bausteinen im Tandembetrieb 32-Bit-Daten verarbeitbar sind.

□

Wilfried Taetow

CMOS-Bausteine für mikroprogrammierbare Signalprozessoren

2. Teil

Für den Aufbau von digitalen Signalprozessoren stehen jetzt auch Funktionsblöcke zur Verfügung, die in CMOS-Technologie gefertigt sind. Diese eignen sich für die Anwendung in mikroprogrammierbaren Systemen zur Signalverarbeitung, die in Hinblick auf Geschwindigkeit und Leistungsfähigkeit Konzepte mit bipolaren Bausteinen sogar noch übertreffen. Nachdem im ersten Teil dieses Beitrages das Familienkonzept der Word-Slice-Bausteine, die verwendete Technologie sowie der Mikroprogramm-Sequenzer ADSP-1401, der universelle Adreßgenerator ADSP-1410 und die arithmetisch-logische Schiebeeinheit ADSP-1201 vorgestellt wurden, zeigt der vorliegende abschließende Teil den erweiterten Multiplizierer-Akkumulator ADSP-1101, den Gleitkomma-Multiplizierer ADSP-3210 und die Gleitkomma-ALU ADSP-3220.

„big MAC“

Familienmitglied Nr. 4 ist der ADSP-1101, ein erweiterter Multiplizierer-Akkumulator (EMAC: „Enhanced Multiplier Accumulator“).

Er ist ebenso wie ALSU ein Dreitor-Baustein, der daraufhin optimiert ist, die sonst notwendige externe Hardware auf ein Minimum zu reduzieren (*Bild* 5). Dieser Chip in einem 96poligen Pin-Grid-Gehäuse besitzt eine größere Leistungsfähigkeit als alle früheren Lösungen. Seine charakteristischen Eigenschaften sind seine Eingangsregisterstruktur, seine anpassungsfähige Architektur, seine extensiven internen Busleitungen und seine Rundungs- und Schiebemöglichkeiten. Die X- und Y-Eingänge des EMAC besitzen je zwei unabhängig adressierbare Register. Diese Speicher auf dem Chip sparen Bus-Zugriffszeit in Anwendungen, die mit doppelter Genauigkeit, oder komplexen Zahlen arbeiten. Außerdem können beide Eingänge je zwei Zahlen in einem einzigen Taktzyklus einlesen.

Der 16×16-Bit-Parallelmultiplizierer des EMAC mit seinen zwei zusätzlichen je 40 Bit breiten Akkumulatoren verarbeitet 2-Komplement- und reine Betragszahlen sowohl untereinander als auch gemischt. Eine Format-Adjust-Vorrichtung erlaubt die Eliminierung des redundanten Vorzeichenbits bei 2-Komplement-Produkten durch Linksverschiebung. Die Kombination all dieser Fähigkeiten führt zu einer enorm verbesserten Durchsatzrate bei so bekannten Algorithmen wie der FFT.

Die zwei 40 Bit breiten Akkumulatoren des EMAC sind in je zwei 16-Bit- und ein 8-Bit-Register aufgeteilt. Die Breite von 40 Bit ist besonders bei solchen Algorithmen vorteilhaft, die eine Vielzahl von Akkumulationen enthalten wie z.B. lange FIR-Filter. Zwischensummen dürfen hier ohne weiteres bis zu 40 Bit groß werden. Das Vorhandensein von zwei Akkumulatoren vereinfacht enorm alle Operationen mit komplexen Zahlen. Sie können aber auch dazu benutzt werden, Zwischenergebnisse abzuspeichern. Weiterhin hat jeder Akkumulator ein Überlauf-Flag, das gesetzt wird, sobald das signifikante Wort in das 8-Bit-Extension-Register hineinwächst. Die interne Busstruktur des EMAC ist ungewöhnlich flexibel. So ermöglicht sie z.B. das vorherige Laden von Daten vom X-Eingang in ein Y-Eingangsregister und das vorherige Laden des Akkumulators vom Y-Eingang. Dieses kann dazu benutzt werden, eine Drei-Operanden-Rechnung wie D: = A × B + C in einem einzigen Zyklus durchzuführen. Eine andere signifikante Eigenschaft dieser internen Busstruktur ist der Rückkopplungspfad, auf dem akkumulierte Resultate zum Y-Eingang transferiert werden können, um z.B. polynomale Reihenentwicklungen für transzendentale Funktionen durchzuführen. Der 16 Bit breite Ausgang dieses Bausteins arbeitet mit einer 50-ns-Taktrate, um beide Anteile des 32-Bit-Produkts in einem 100-ns-Systemtakt ausgeben zu können. Der Ausgang des Akkumulators kann zuvor noch um 1 Bit nach rechts oder links verschoben werden. Linksverschiebungen sind nützlich zur Normierung von Zweier-Komplementzahlen; Rechtsverschiebungen dienen zur Formatierung von Block-Fließkomma-Zahlen.

Natürlich kann der EMAC auf Wunsch auf das 14., 15. oder 16. Bit runden, je nachdem, welche Verschiebungen durchgeführt werden. Alle Rundungen sind Offset-frei, d.h. liegt die zu rundende Zahl genau in der Mitte, so wird nach einem statistischen Verfahren gleich oft auf- und abgerundet. Diese Eigenschaft macht sich gerade bei Algorithmen wie der FFT positiv bemerkbar. Rundungsschemata mit Offset, wie man sie in Multiplizierern der ersten Generation findet, täuschen nämlich

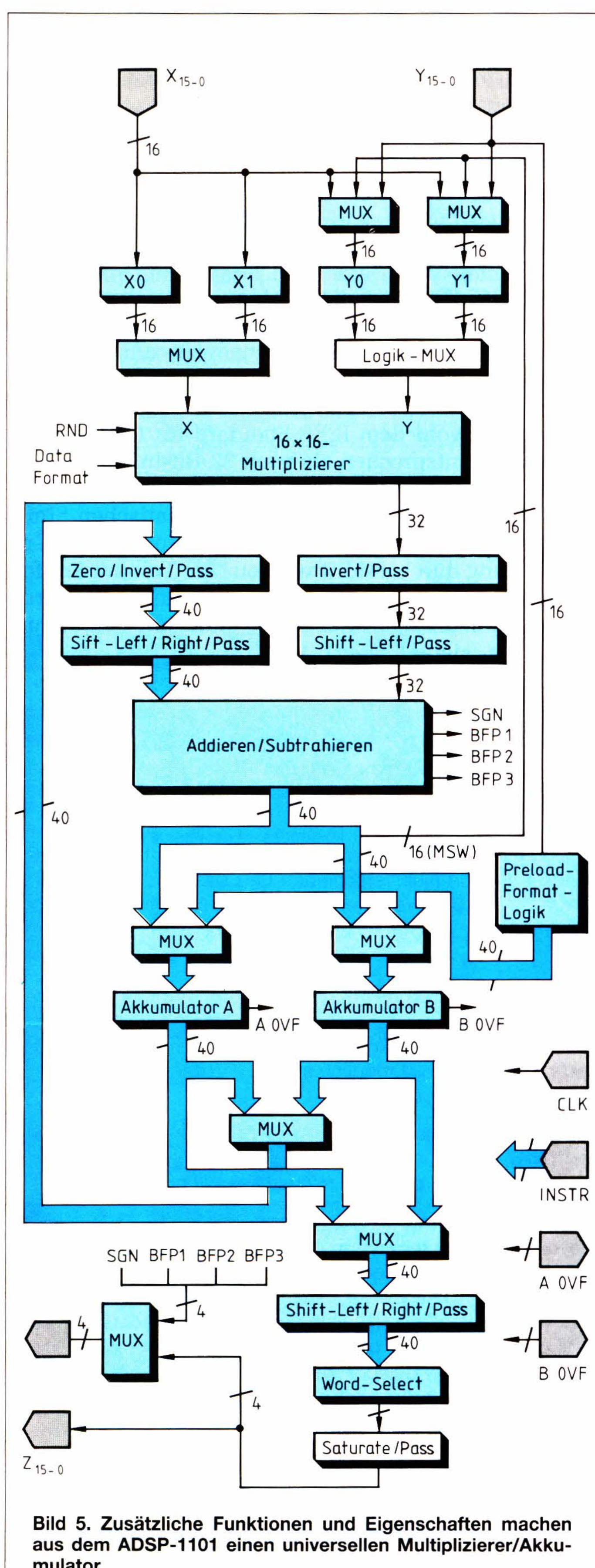


Bild 5. Zusätzliche Funktionen und Eigenschaften machen aus dem ADSP-1101 einen universellen Multiplizierer/Akkumulator

einen nicht vorhandenen Gleichspannungsanteil bei der Spektralanalyse vor. Die eingebaute Sättigungslogik kann in Verbindung mit dem Überlauf-Flag das gefürchtete „Wrap-Around" von Zweier-Komplementzahlen verhindern. Wächst z.B. eine positive Zweier-Komplementzahl durch mehrfache Akkumulation über das vereinbarte Zahlenformat (32 Bit) hinaus und werden die acht Zusatzbit nicht benutzt, so kehrt sie sich in eine negative Zahl um. Mit seiner bedingungsabhängigen Sättigungslogik erzeugt der Chip automatisch den positiven bzw. negativen Bereichsendwert und verhindert so Großsignalschwingungen in rekursiven Berechnungen.

Gleitkomma auch für Signalverarbeitung

War die Gleitkommaverarbeitung bisher hauptsächlich auf Universalrechner beschränkt, so dringt sie nun mehr und mehr auch in die digitale Signalverarbeitung vor. Überlaufprobleme, Rundungsrauschen, Grenzzyklen, zu geringe Dynamik sind einige der Sorgen, die den Entwicklern von digitalen Signalverarbeitungssystemen immer wieder Kopfzerbrechen bereiten. Während z.B. ein Filteralgorithmus in Fortran schnell entworfen ist und in der ersten Simulation hervorragende Eigenschaften zeigt, bringt die nächste Simulationsstufe unter Berücksichtigung der zur Verfügung stehenden Hardware meist die große Ernüchterung. Jetzt wird um jedes Bit gekämpft und es sind dem Einfallsreichtum der Entwickler keine Grenzen gesetzt, nur um möglichst nahe an das mathematische Ideal heranzukommen. Ein Großteil der Entwicklungsarbeit wird darauf verwendet, die Auswirkungen der Rechenungenauigkeit verfügbarer Hardware-Bausteine so gering wie möglich zu halten.

Einer dieser Tricks ist z.B. die Verwendung des Block-Fließkomma-Formats (siehe Teil 2), das sich gerade in der FFT einer großen Beliebtheit erfreut. Die Effektivität dieser Zahlendarstellung ist allerdings stark abhängig von der Art des verwendeten Algorithmus; sie kann daher nicht als universelles „Allheilmittel" angesehen werden.

Warum verwendet man nicht Gleitkommazahlen, wie dies jeder Personal Computer bereits kann? Der Grund liegt natürlich in der Verfügbarkeit geeigneter Hardware-Bausteine. Aus heutiger Sicht bieten sich dem Entwickler drei Möglichkeiten:

- Man konfiguriere auf einer oder mehreren Platinen jede Menge LSI-, MSI- und SSI-Logikbausteine, wie es in vielen Array-Prozessoren gang und gäbe ist. Obschon solche Systeme sehr leistungsfähig und schnell sein können, sind sie doch meist recht groß und vor allem teuer.
- Man übertrage Gleitkommarechnungen arithmetischen Coprozessoren, die zusammen mit Mikroprozessoren arbeiten. Sie beherrschen eine Vielzahl von Funktionen einschließlich Gleitkommarechnung. Unglücklicherweise sind solche Coprozessoren für die meisten hier interessierenden Anwendungen zu langsam und zudem auf spezielle µP-Bussysteme zugeschnitten.

– Man benutze einen der wenigen z. Z. verfügbaren LSI-Gleitkommabausteine. Das Problem hier ist die begrenzte Flexibilität und bei fast allen die Einschränkung auf einfache Genauigkeit. Diese Bausteine sind verglichen mit Bit-Slice-Systemen immer noch zu langsam und besitzen zudem meist zahlreiche interne Pipeline-Register, was die Software-Entwicklung und Steuerung verkompliziert.

Vielseitig und doch schnell

Die beiden letzten Mitglieder der Word-Slice-Familie, der Gleitkommamultiplizierer ADSP-3210 und die Gleitkomma-ALU ADSP-3220 kombinieren die Vielseitigkeit und Geschwindigkeit von Prozessoren auf MSI-Basis mit einer Vielzahl von Funktionen und Datentypen von arithmetischen Coprozessoren. Als Resultat können dann Systeme aufgebaut werden, die, was die Gleitkommarechnung angeht, an die Leistungsfähigkeit eines Großrechners heranreichen, mit Mitteln, die dem Aufbau von Mikro-Computern ähneln.

Ein einziger numerischer Prozessor, der basierend auf diesem Chip-Paar aufgebaut ist, kann 32-Bit-Gleitkommazahlen mit einer Durchsatzrate von 10 MFLOPS berechnen (1 MFLOP = 1 Million Floating Point Operations per Second).

64-Bit-Gleitkommazahlen mit doppelter Genauigkeit werden mit 2 MFLOPs und 32-Bit-Festkommazahlen mit 10 MHz Durchsatzfrequenz berechnet. Zum Vergleich seien die Durchsatzraten zweier bekannter Computer angegeben:

VAX-11/780: 0,3 MFLOPS (64 Bit)
CRAY-1: 80 MFLOPS (64 Bit)

Da beide CMOS-Bausteine intern nicht dynamische, sondern statische Logik verwenden, können sie an jede Geschwindigkeit bis 10 MHz angepaßt und somit an existierende Bussysteme angeschlossen werden.

Von großer Bedeutung ist die Tatsache, daß beide Bausteine sowohl dem IEEE-Standard für Gleitkommaprozessoren entsprechen als auch 32-Bit-Integer-Zahlen verarbeiten können. Dies macht sie interessant für CAD-Entwurfssysteme, wo neben der mathematischen Simulation auch umfangreiche Adreßberechnungen zur Grafikerzeugung durchgeführt werden. Als Teil der Konformität unterstützen beide Bausteine alle fünf standardisierten Gleitkommazahlentypen und alle vier Rundungsarten (siehe *Kasten*).

Der IEEE-Standard 754

Der IEEE-Standard setzt eine Norm für binäre Gleitkomma-Arithmetik. Er spezifiziert Datenformate, Operandentypen, Operationen, Rundungsmodi und Ausnahmebehandlung.

Einfache Genauigkeit:
Die Datenwortbreite ist 32 Bit, die wie folgt aufgeteilt ist: Das erste Bit repräsentiert das Vorzeichen, die nächsten 8 Bit sind der Exponent und die verbleibenden 23 Bit repräsentieren die Mantisse.

Doppelte Genauigkeit:
Die Wortbreite ist 64 Bit, wovon 1 Bit für das Vorzeichen, 11 Bit für den Exponenten und 52 Bit für die Mantisse vorgesehen ist.

Vier verschiedene Operandentypen sind spezifiziert:
1. Normierte Zahlen
Sie repräsentieren sozusagen den Normalfall und werden wie folgt dargestellt:

Format	Wert	Bereich
V $E_7 \ldots E_0$ $M_{22} \ldots M_0$ 1 8 23	$(-1)^V \cdot 2^{E-127} \cdot 1,M$	$1 \leq E \leq 254$
V $E_{10} \ldots E_0$ $M_{51} \ldots M_0$ 1 11 52	$(-1)^V \cdot 2^{E-1023} \cdot 1,M$	$1 \leq E \leq 2046$

Der Wert des Exponenten besitzt die Basis 2 und einen Offset von 127 bzw. 1023. Die Wortbreite der Mantisse wird implizit um ein Bit („Hidden Bit") vergrößert und stellt einen Dezimalbruch mit einer Vorkommastelle dar.

2. Denormierte Zahlen
Sie repräsentieren Zahlen, deren Wert kleiner als der minimale normiert darstellbare Wert ist. Der Exponent ist Null, die Mantisse ist von Null verschieden und das „Hidden Bit" ist ebenfalls Null. Der Wert einer solchen Zahl ergibt sich aus:
$(-1)^V \cdot 2^{-126} \cdot 0,M$ bzw.
$(-1)^V \cdot 2^{-1022} \cdot 0,M$

3. Vorzeichenbehaftete Null
Die Null wird am Exponenten von Null und einer Mantisse von Null erkannt. Das Vorzeichen-Bit bleibt jedoch gültig, so daß Werte von +0 und –0 möglich sind.

4. Unendlich
Plus und minus Unendlich haben einen Exponenten von 255 bzw. 2047 und eine Basis von Null. Das Vorzeichen unterscheidet zwischen plus und minus Unendlich.

5. Nicht-Zahlen (NAN – „Not A Number")
Wie der Name bereits vermuten läßt, handelt es sich hier nicht um Zahlen im eigentlichen Sinn, sondern vielmehr um ein spezielles Datenformat, das gewöhnlich als Flag im Datenfluß verwendet wird. Das Format dieser Nicht-Zahl ist ein Exponent von 255 bzw. 2047, aber mit einer Mantisse ungleich Null. Zum Beispiel werden solche Nicht-Zahlen zur Anzeige von nicht definierten Operationen (0 ×) verwendet.

Der IEEE-Standard verlangt vier verschiedene Rundungsarten:
- Rundung auf die nächstgelegene Zahl
- Rundung in Richtung Null
- Rundung in Richtung plus Unendlich
- Rundung in Richtung minus Unendlich

Beide Bausteine sind zudem voll kompatibel mit den Ausnahme-Behandlungsmethoden gemäß dem Standard. Tritt ein Ausnahmefall auf, so geben die Chips einen oder mehrere Status-Flags aus, die den entsprechenden Fall spezifizieren, so daß die steuernde Software das Resultat korrekt interpretieren kann.

Obwohl die Verarbeitung von denormalisierten Zahlen selbstverständlich möglich ist, mußte aus Rücksicht auf die Ausmaße der Chips eine Ausnahmeregelung getroffen werden. Während der ALU-Baustein ADSP-3220 solche Zahlen ohne äußere Hilfe verarbeiten kann, ist der Multiplizierer ADSP-3210 auf die Unterstützung der ALU angewiesen. Der Grund liegt darin, daß auf dem Multiplizierer-Chip kein Platz mehr ist, um den sonst notwendigen Barrel-Shifter unterzubringen. Will man ihn trotzdem zur Verarbeitung von denormalisierten Zahlen einsetzen, so wird folgender „Trick" angewandt:

Zunächst wird mit Hilfe der ALU die Mantisse der denormalisierten Zahl normiert. Es besteht eine sogenannte „verdrehte Zahl" (Wrapped Number), die eine normierte Mantisse, aber einen negativen Exponenten besitzt, d. h. der Exponent wurde durch die Normierung soweit vermindert, daß er nun eine negative Zwei-Komplementzahl darstellt. Der Offset des Exponenten bleibt jedoch nach wie vor 127. Der Multiplizierer kann nun diese verdrehten Zahlen ohne weiteres verarbeiten, da sein Exponentenaddierer sowohl positive als auch negative Zahlen akzeptiert. Das Produkt wird nun wieder an die ALU zurückgegeben und gegebenenfalls wieder in eine denormalisierte Zahl verwandelt. Beide Chips haben entsprechende Steuer- und Kontrollein- und -ausgänge, so daß dieser ganze Vorgang automatisch ablaufen kann. Ist die Zeitverzögerung bei der Behandlung von denormierten Zahlen nicht akzeptabel, so läßt sich der Multiplizierer in den „Fast-Modus" schalten. Wird jetzt eine Zahl als denormalisiert erkannt, wird sie automatisch zu Null gesetzt. Dies entspricht zwar nicht dem IEEE-Standard, kann aber in zeitkritischen Anwendungen als akzeptable Näherung angesehen werden.

Beide Chips haben intern das gleiche Bus-Interface, das sich für einen großen Bereich von Anwendungen hervorragend eignet. Jeder Baustein besitzt vier unabhängige 32-Bit-Register, aufgeteilt in A- und B-Gruppen. Der Datentransport zu und von den Chips geschieht mit der zweifachen Taktrate (50 ns), so daß die Verarbeitungsgeschwindigkeit nicht durch E/A-Transfers verlangsamt wird. In jedem Systemzyklus von 100 ns werden jeweils ein A- und B-Register als Quelle für die entsprechende arithmetische Operation ausgewählt. Während des Rechenvorgangs können die beiden anderen A- und B-Register bereits neu geladen werden.

Beide Bausteine besitzen nur eine einzige Pipeline-Stufe, was die Erstellung des Mikrocodes wesentlich vereinfacht und „Verstrickungen" im Signalfluß verhindert.

Der Kern des Gleitkommamultiplizierers ADSP-3210 (*Bild 6*) ist das parallele 32×32-Bit-Multiplikationsnetzwerk mit einer Verarbeitungszeit von 100 ns. Diese Geschwindigkeit konnte durch eine modifizierte Booth-/Wallace-Struktur erreicht werden. Dies bedeutet, 32-Bit-Fest- und Gleitkommaoperanden werden mit einer effektiven Durchsatzrate von 10 MOPS bzw. 10 MFLOPS zu 64-Bit-Festkomma- oder 32-Bit-Gleitkommaprodukten verarbeitet. Vom Eingang bis zum Ausgang benötigt die Berechnung natürlich aufgrund des Pipeline-Registers zwei Zyklen. Während des ersten Zyklus wird das Produkt berechnet, und im zweiten Zyklus wird das Resultat gerundet, formatiert und ausgegeben.

Die Multiplikation von doppelt genauen Gleitkommazahlen wird mit einer effektiven Durchsatzrate von 2 MFLOPS auf Befehl automatisch und ohne äußere Steuerung ausgeführt.

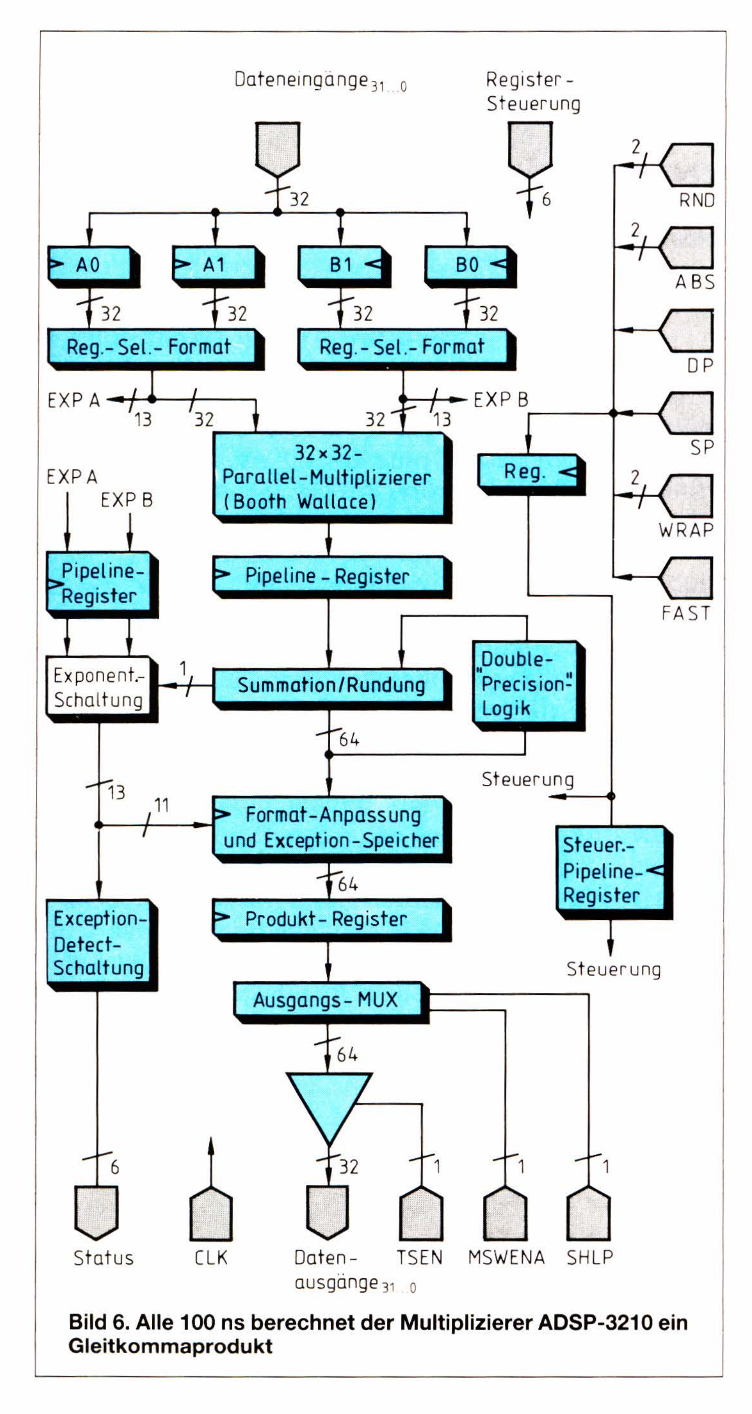


Bild 6. Alle 100 ns berechnet der Multiplizierer ADSP-3210 ein Gleitkommaprodukt

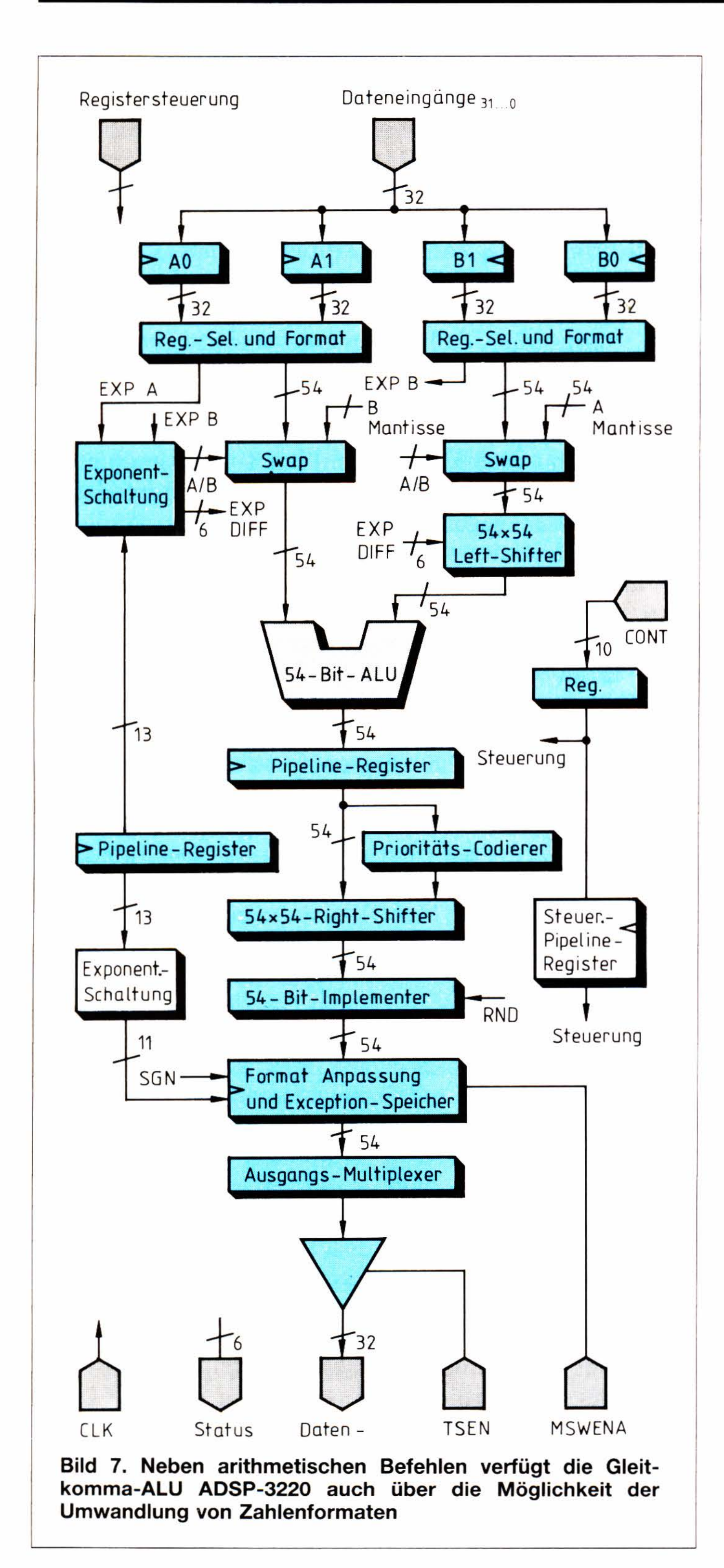


Bild 7. Neben arithmetischen Befehlen verfügt die Gleitkomma-ALU ADSP-3220 auch über die Möglichkeit der Umwandlung von Zahlenformaten

Das letzte Mitglied unserer Word-Slice-Familie ist die Gleitkomma-ALU ADSP-3220. Ebenso wie der Multiplizierer verarbeitet auch sie drei Datenformate (*Bild 7*).

Die Durchsatzrate beträgt für 32-Bit-Fest- oder Gleitkommazahlen 10 MOPS bzw. 10 MFLOPS, während die Verarbeitung von Gleitkommazahlen mit doppelter Genauigkeit eine Durchsatzrate von 5 MFLOPS erreicht.

Der ADSP-3220 führt alle gängigen arithmetischen und logischen Operationen aus wie Addition, Subtraktion, logische Verknüpfungen und Umwandlungen zwischen Fest- und Gleitkommazahlen. Außerdem führt er die bereits erwähnte Zahlenverdrehung zur Unterstützung des Multiplizierers durch.

Während dieser 1. Teil des Beitrags der Vorstellung der einzelnen Bausteine der Word-Slice-Familie gewidmet war, werden im 2. Teil drei verschiedene Prozessorarchitekturen vorgeschlagen und am Beispiel einer 1024-Punkte-FFT auf ihre Leistungsfähigkeit untersucht.

Dipl.-Ing. Wolfgang Wirwahn
Cand.-Ing. Reinhold Ludwig

Signalprozessor als Funktionsmodul im Mikrocomputersystem

In der Praxis beschränken sich digitale Filter meistens auf mikroprogrammierte Hardwarestrukturen, womit der Flexibilität enge Grenzen gesetzt sind. Für den Entwurf eines universellen digitalen Signalprozessors bietet sich deshalb die Einbindung in ein Rechnersystem an. Nachfolgend wird ein Konzept für die Anwendung im Niederfrequenzbereich vorgestellt, bei dem ein Digitalfilter sowie die zugehörige PCM-Einheit in einem herkömmlichen Z80-Mikroprozessorsystem eingebunden ist. Hierbei ergab sich, daß der Anwendungsspielraum des ursprünglich als nichtrekursives Digitalfilter aufgebauten Gerätes sich erweitern ließ.

1 Allgemeine Anforderungen

Für die digitale Signalverarbeitung im Niederfrequenzbereich ist wegen des gewünschten Dynamikumfanges von bis zu 90 dB eine Verarbeitung mit einer Wortbreite von 14...16 Bit erforderlich. Die Abtastfrequenz soll 50 kHz betragen, was einer Zeitspanne zwischen zwei Abtastwerten von 20 µs entspricht. Auf diesen Wert ist der verfügbare Verarbeitungszeitraum eines digitalen Filters festgelegt. Im Gegensatz zur mikroprogrammierten Hardwarestruktur, bei der die Einstellmöglichkeiten außerhalb des Systems liegen (*Bild 1*), soll hier ein Konzept vorgestellt werden, welches ein vollständig von einem Mikroprozessor gesteuertes digitales Verarbeitungssystem realisiert (*Bild 2*). Die Einstellmöglichkeiten liegen hier innerhalb des Systems, da alle Baugruppen in das Mikroprozessorsystem einbezogen sind.

Insbesondere soll im folgenden gezeigt werden, daß es trotz der zu verarbeitenden Wortbreite von 16 Bit möglich ist, die gezeigte Struktur mit einem normalen 8-Bit-Mikroprozessor (Z80) zu realisieren.

2 Systemaufbau

Bild 3 zeigt das gesamte digitale Verarbeitungssystem einschließlich des steuernden Mikroprozessors als Blockschaltung.

Neben den für den Aufbau eines einfachen µP-Systems hinreichend bekannten Einheiten CPU, RAM, EPROM und E/A sind drei weitere Baugruppen erforderlich, die die eigentliche Signalverarbeitung durchführen:

KE – Codiereinheit
DSE– Datentransfer- und Synchronisiereinheit
FDE– Filter- und Decodiereinheit

Die spezifischen Merkmale dieser Funktionsblöcke sind:

Codiereinheit KE

Die KE erzeugt aus dem analogen Niederfrequenzsignal das zur Digitalfilterung erforderliche PCM-codierte Signal. Diese Baugruppe ist als E/A-Port adressierbar und besteht im wesentlichen aus einer Sample-and-Hold-Schaltung mit nachgeschaltetem Analog-Digital-Umsetzer.

Datentransfer- und Synchronisiereinheit DSE

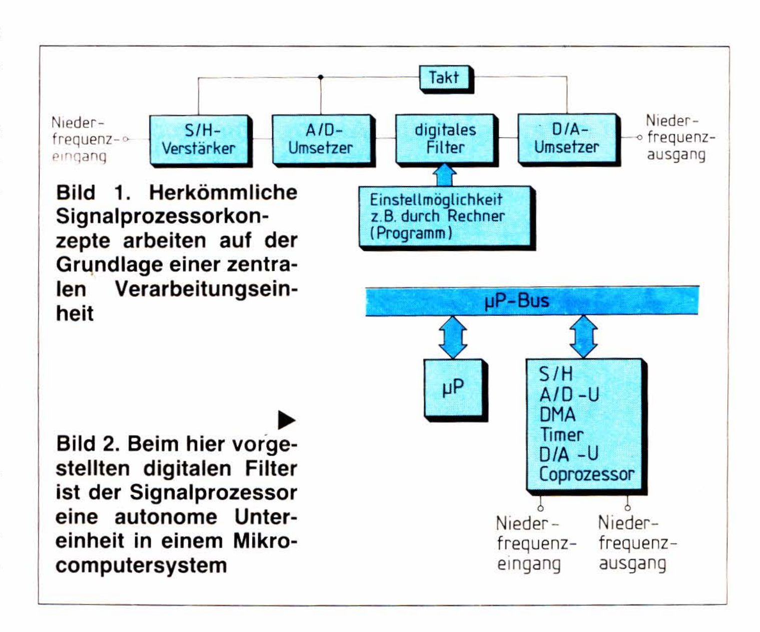


Bild 1. Herkömmliche Signalprozessorkonzepte arbeiten auf der Grundlage einer zentralen Verarbeitungseinheit

▶ **Bild 2. Beim hier vorgestellten digitalen Filter ist der Signalprozessor eine autonome Untereinheit in einem Mikrocomputersystem**

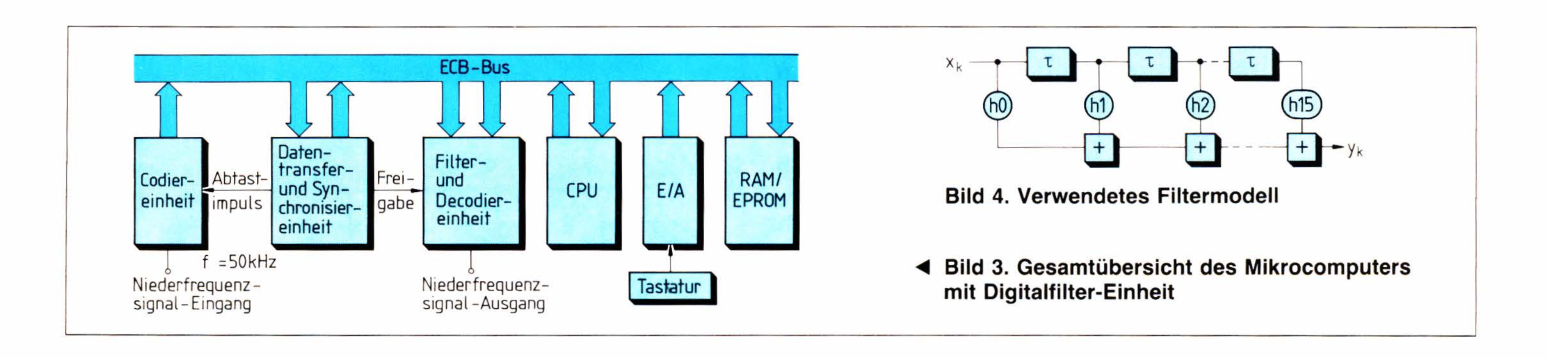


Bild 4. Verwendetes Filtermodell

◀ **Bild 3. Gesamtübersicht des Mikrocomputers mit Digitalfilter-Einheit**

Die DSE enthält für den schnellen Datentransfer zwischen Codierer und Filter einen DMA-Controller sowie einen CTC (Counter Timer Controller) zur Erzeugung von phasenstarren Taktsignalen, die eine Synchronisation zwischen Codierer, Filter und Decodierer herstellen.

Filter- und Decodiereinheit FDE

Die letzte zu betrachtende Baugruppe FDE ist hier nur als Baueinheit, nicht aber als Funktionseinheit aufzufassen. Sie beinhaltet im ersten Teil den eigentlichen digitalen Signalprozessor in der Form eines nichtrekursiven Digitalfilters. Der Zusammenhang zwischen den Eingangswerten x_k, x_{k-1}, x_{k-2}... und dem Ausgangswert y_k wird durch die Gleichung

$$y_k = \sum_{m=0}^{M} h_m \, x_{k-m} \qquad (1)$$

beschrieben und kann gemäß *Bild 4* dargestellt werden. Die Größen h_m sind hierbei konstante Bewertungskoeffizienten. Dieser mathematische Zusammenhang ist im vorliegenden System durch Einsatz eines schnellen 16x16-Bit-Multiplizierer-Akkumulators sowie eines Daten-, Koeffizienten- und Mikroprogramm-RAMs (Mikroprogramm für Multiplikation mit anschließender Addition) realisiert.

Koeffizienten-RAM und Mikroprogramm-RAM sind doppelt ausgeführt, so daß ein Austausch der Filterparameter durch Umschalten der RAM-Sätze zum geeigneten Zeitpunkt ohne Störung der Abarbeitung im Filter möglich ist. Im zweiten Teil der FDE wird der Filterausgangswert mit einem Digital-Analog-Umsetzer decodiert und in ein analoges Signal umgesetzt.

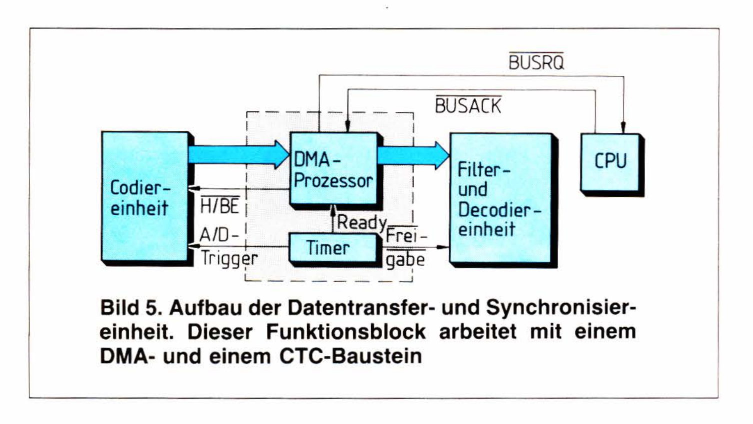


Bild 5. Aufbau der Datentransfer- und Synchronisiereinheit. Dieser Funktionsblock arbeitet mit einem DMA- und einem CTC-Baustein

3 Realisierung

Der in der Codiereinheit KE verwendete Baustein A/D/A/M-724 der Firma Analogic enthält die für die PCM-Codierung notwendigen Module: Sample-and-Hold-Verstärker und Analog-Digital-Umsetzer. Er gestattet eine Signalauflösung von 14 Bit bei einer maximalen Umsetzzeit von 7 µs. Unter Berücksichtigung einer Erholzeit von 3 µs der Sample-and-Hold-Schaltung ergibt sich eine maximale Abtastfrequenz von 100 kHz. Die Umsetzung wird durch eine von der DSE bereitgestellte Impulsfolge „A/D-Trigger“ eingeleitet. Sie definiert die hier auf 50 kHz festgelegte Abtastfrequenz. Die Einstellung anderer Frequenzen ist möglich. Mit dem zusätzlich von der DSE bereitgestellten Signal „HIBE“ (High Byte Enable) werden entweder die höherwertigen 6 Bit oder die niederwertigen 8 Bit der 14 Multiplex-Datenleitungen des Codierers selektiert. Die dadurch notwendige sequentielle Abfrage der Daten in zwei Zugriffen reduziert den Informationsdurchsatz, ist aber aufgrund der 8-Bit-Struktur des Systemdatenbusses und des DMA-Prozessors unvermeidbar.

Die Datentransfer- und Synchronisiereinheit DSE erfüllt die folgenden zwei Aufgaben:

A) Bereitstellung der für die Synchronisation notwendigen Signale durch einen Timer

B) Datentransfer von der KE zu der FDE mit einem DMA-Prozessor.

Bild 5 zeigt die prinzipielle Funktionsweise.

Der Timer leitet mit der Ausgabe der Impulsfolge „A/D-Trigger“ die Umsetzung in der KE ein. Mit einem Zeitversatz, der mit 8 µs etwas über der Umsetzdauer des Codierers liegt, erfolgt die Aktivierung des DMA-Prozessors über die Ready-Leitung. Als Reaktion hierauf fordert dieser mit $\overline{\text{BUSRQ}}$ die Übernahme der Bussteuerung von der CPU an. Wird CPU-seitig die Anforderung mit $\overline{\text{BUSACK}}$ quittiert, können unter Kontrolle des DMA-Prozessors byteseriell die höherwertigen 6 Bit (High-Byte) und die niederwertigen 8 Bit (Low-Byte) des Codierers in die Speicher der FDE transferiert werden. Sind die Daten mit Hilfe der HIBE-Leitung in der zeitlichen Reihenfolge High-Byte und Low-Byte in die entsprechenden Speicherzellen der FDE geladen, gibt der DMA-Prozessor infolge der nun inaktiven Ready-Leitung die Buskontrolle an die CPU zurück. Schließlich

erteilt der Timer mit einem Zeitversatz von 6 µs gegenüber der Aktivierung der Ready-Leitung bzw. 14 µs gegenüber „A/D-Trigger", den Freigabeimpuls für die Verarbeitung in der FDE. Der sich mit der Abtastfrequenz von 50 kHz wiederholende Gesamtvorgang ist als Zeitdiagramm in *Bild 6* qualitativ dargestellt.

Kern der PCM-Übertragungsstrecke ist die in *Bild 7* gezeigte Filter- und Decodiereinheit FDE. Die Hauptbestandteile sind dabei:

1) die doppelt ausgeführten Koeffizienten- und Statusinformationsspeichereinheiten
2) die Taktsteuerung für den Multiplizierer und Address-Pointer
3) die Speicher für die von der DSE transferierten codierten Abtastwerte
4) der Address-Pointer für die Adressierung der unter 1) und 3) genannten Speicher
5) der Multiplizierer für die Signalverarbeitung
6) die Decodierung mit D/A-Umsetzer.

Vor Inbetriebnahme der FDE ist mindestens eine der beiden Koeffizientenspeichereinheiten mit einer Organisation von 16 x 16 Bit zu laden. Jedem Koeffizienten-RAM fest zugeordnet ist ein Statusinformations-RAM (Organisation: 16 x 4 Bit) für die Mikroprogrammierung des Rechenwerkes. Diese Speichereinheit muß gemeinsam mit den Koeffizienten geladen werden. Abschließend wird die Filtertiefe vorgegeben, d. h. die Anzahl der an der Filterung beteiligten Koeffizienten. Der die Filtertiefe bestimmende Faktor M in Gleichung (1) kann hierbei von 0...15 beliebige ganzzahlige Werte annehmen. Die Speicherunterteilung in zwei getrennte Einheiten erlaubt den Austausch sämtlicher am Filtermodell beteiligter Koeffizienten in Echtzeit, d. h. zwischen zwei Abtastzeitpunkten.

Der Speicher für die codierten Signale übernimmt die durch den DMA-Prozessor transferierten Signalwerte. Die Speichertiefe wird wie bei den Koeffizienten-RAMs über die Filtertiefe M eingestellt. Das Abspeichern der transferierten Daten erfolgt rotierend, d. h. der jeweils aktuelle überschreibt den ältesten Signalwert.

Nach Beendigung des DMA-Transfers erteilt die DSE den Freigabeimpuls, der die Taktsteuerung veranlaßt, drei gegeneinander zeitverschobene Impulsfolgen der Anzahl M+1 auszugeben. Die erste Taktleitung steuert den Address-Pointer, die beiden anderen sind für den Multiplizierer bestimmt.

Der Address-Pointer übernimmt die zeitlich richtige Zuordnung zwischen den codierten Signalwerten und den jeweiligen Koeffizienten. Nach jedem Freigabeimpuls aus der DSE adressiert dieser in schneller Folge (Periode = 375 ns) sämtliche relevanten Speicherzellen und steuert dadurch den Datenfluß zum Multiplizierer.

Hauptbestandteil für den digitalen Verarbeitungsprozeß ist der Multiplizierer und Akkumulator TDC 1010J der Firma TRW. Eingangsseitig übernimmt diese Arithmetikeinheit die codierten Signale sowie die bereitgestellten Koeffizienten, um zu einem späteren Zeitpunkt das Verarbeitungsergebnis an den Decodierer auszugeben. Die im Filtermodell (Bild 4) vorkommenden Operationen der Multiplikation und Addition werden von dem Baustein unter Kontrolle eines Mikroprogramms ausgeführt. Mit jedem Koeffizienten wird ein Mikrobefehl als 4-Bit-Statusinformationswort bereitgestellt. Dieses definiert neben den Rechenoperationen der Multiplikation, Addition oder Subtraktion die Art der Codierung, z. B. Zweierkomplement. Diese Form der Bausteinsteuerung erlaubt es, zu jedem einzelnen Abtastwert und Koeffizient die dazugehörige Verarbeitung zu wählen. Es ist deshalb auch möglich, neben dem in Bild 4 gezeigten Modell andere Formen von Filtermodellen aufzubauen. Die gemeinsame Übernahme der Abtastwerte, Koeffizienten und Mikrobefehle in die Register der Arithmetikeinheit wird durch eine von der Taktsteuerung erzeugte Impulsfolge veranlaßt. Eine zweite Impulsfolge liefert den Übernahmetakt in das Ausgangsregister. Sie hat gegenüber der ersten eine Zeitverschiebung von 200 ns. Damit ist sichergestellt, daß dem Multiplizierer genügend Zeit für die Signalverarbeitung zur Verfügung steht, ehe die Rechenergebnisse in das Ausgangsregister übernommen werden. In Abhängigkeit von der Filtertiefe M werden M+1 Abtastwerte und Koeffizienten sukzessiv mit der ersten Impulsfolge in den Multiplizierer übernommen und verarbeitet, bevor die zweite Impulsfolge die jeweils aufaddierten Zwischenergebnisse in das Ausgangsregister

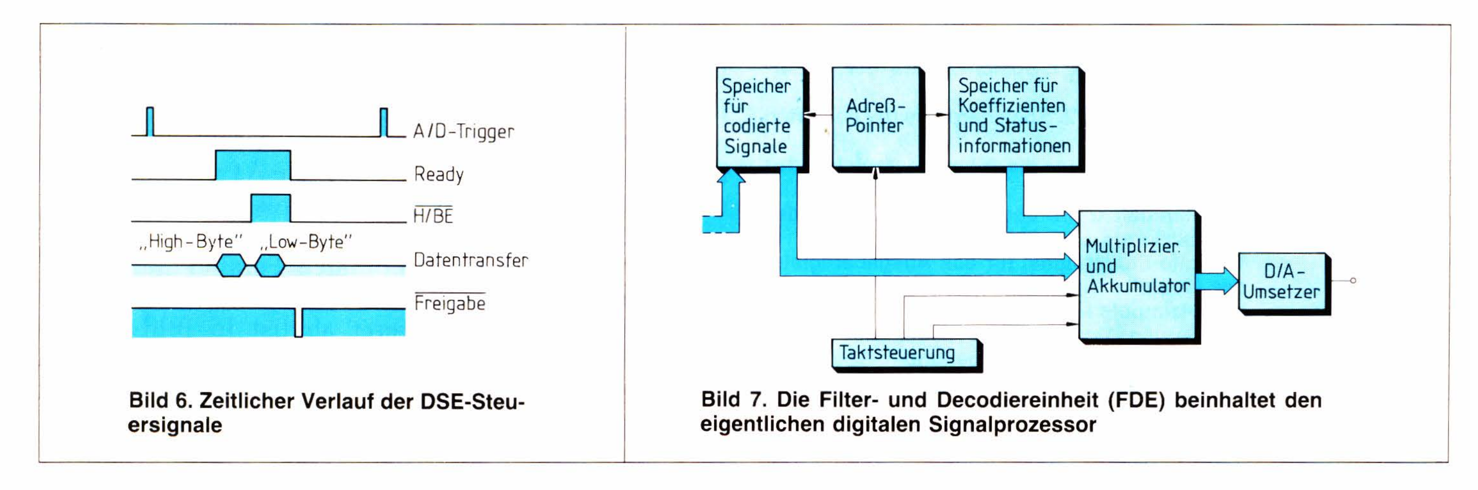


Bild 6. Zeitlicher Verlauf der DSE-Steuersignale

Bild 7. Die Filter- und Decodiereinheit (FDE) beinhaltet den eigentlichen digitalen Signalprozessor

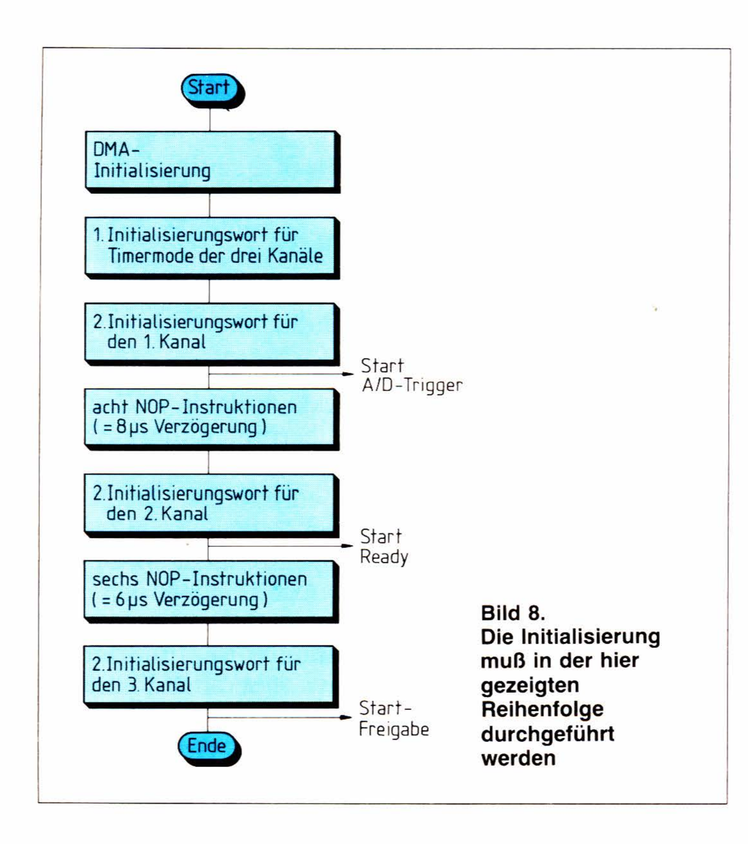


Bild 8. Die Initialisierung muß in der hier gezeigten Reihenfolge durchgeführt werden

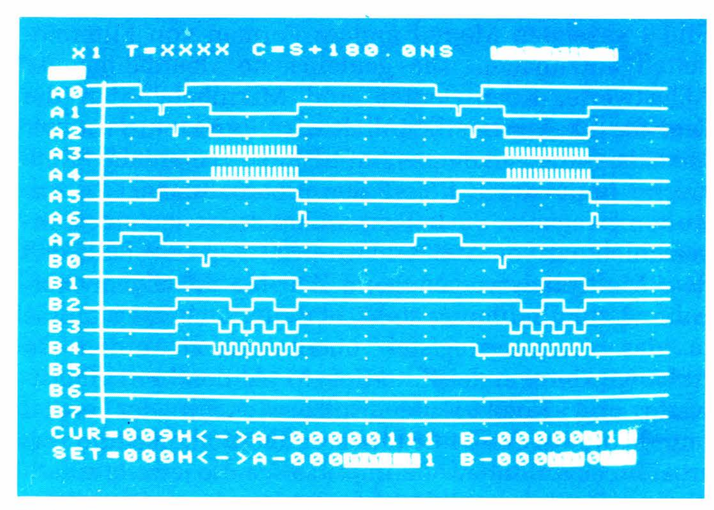


Bild 9. Zeitlicher Verlauf der Steuerimpulse im Gesamtsystem

Es bedeuten:

A0 = $\overline{BAI}$:	**Systembus-Übernahme durch DMA-Prozessor**
A1 = $\overline{CE1}$:	**Chip-Enable des 1. Speicherpaars der codierten Signalwerte**
A2 = $\overline{CE2}$:	**Chip Enable des 2. Speicherpaars der codierten Signalwerte**
A3 = CLK X,Y:	**Übernahmetakt in die Eingangsregister des Multiplizierers**
A4 = CLK P:	**Übernahmetakt in das Ausgaberegister des Multiplizierers**
A5 = $\overline{TSM}$:	**Tri-State-Steuersignal für das Ausgaberegister des Multiplizierers**
A6 = A/D-Trigger:	**Umsetzimpuls für KE**
A7 = $\overline{Ready}$:	**Aktivierungssignal für DMA-Prozessor**
B0 = $\overline{Freigabe}$:	**Startsignal der Filterverarbeitung**
B1 (MSB), B2, B3, B4 (LSB):	**Teil des Address-Pointers (hier: das Daten-RAM adressierend)**
B5, B6, B7:	**ohne Bedeutung**
Zeitmaßstab:	**4 µs zwischen zwei vertikal gepunkteten Linien**

abspeichert. Das nach M+1 Schritten im Ausgangsregister stehende Endergebnis wird dem Decodierer zur Umsetzung bereitgestellt.

Als Decodierer findet ein 16-Bit-D/A-Umsetzer, MP 1926A von Analogic mit 3 µs Einschwingzeit Verwendung. Zur Unterdrückung der am Ausgang des D/A-Umsetzers auftretenden Spannungsspitzen ist ein „Deglitcher" MP 201A derselben Firma nachgeschaltet.

4 Initialisierungssoftware

Die Systeminitialisierung unterteilt sich in die Einzelinitialisierungen für die FDE und DSE.

Bei der FDE werden die Mikroinstruktionen durch direktes Beschreiben einer zu der Filtertiefe M korrespondierenden Anzahl von Speicherzellen geladen. Die Filtertiefe wird durch Ausgabe eines entsprechenden Datenwortes in das als E/A-Port adressierte FDE-Register festgelegt.

Bei der DSE wird zuerst der DMA-Prozessor für den Datentransfer von der KE zur DSE initialisiert. Die vom Timer zu generierenden drei zeitversetzten 50 kHz Takte erreicht man dadurch, daß die betreffenden CTC-Kanäle eine zeitversetzte Initialisierung erfahren. Hierbei werden die Zeitlücken durch Abarbeitung von NOP-Instruktionen überbrückt. Eine NOP-Instruktion gestattet eine Zeitquantisierung von 1 µs bei einem Systemtakt von 4 MHz. *Bild 8* zeigt die DSE-Initialisierung als Flußdiagramm.

5 Zeitverhalten

Bild 9 zeigt das typische Zeitverhalten des Gesamtsystems.

Die PCM-Übertragungsstrecke beginnt die Codierung eines Analogwertes mit A/D-Trigger (Kanal A6). Unter Berücksichtigung der Umsetzzeit wird nach 8 µs die Ready-Leitung am DMA-Prozessor aktiviert (Kanal A7). Mit einer Zeitverzögerung von 1,22 µs gegenüber der positiven Flanke auf der Ready-Leitung übernimmt der DMA-Prozessor die Bussteuerung (Kanal A0). Diese Zeitdauer beträgt 3,36 µs, während der der Abtastwert in die Datenspeicher der FDE eingeschrieben wird (Kanäle A1 und A2). Mit dem 14 µs nach „A/D-Trigger" folgenden Freigabeimpuls (Kanal B0) beginnt die eigentliche Filterverarbeitung. Am Beispiel des Signalpointers (B1...B4) als Teil des Address-Pointers ist die Speicheradressierung dargestellt. Zuerst werden die beiden Datenspeicherpaare aktiviert (Kanäle A1 und A2), bevor der Signalpointer die einzelnen Adressen generiert. Die zugehörigen Speicherzelleninhalte werden mit CLK X,Y in den Multiplizierer übernommen. Parallel hierzu läuft der Vorgang der Koeffizientenadressierung mit Übernahme in den Multiplizierer ab (ist in Bild 9 nicht dargestellt). CLK P (Kanal A4) lädt die Verknüpfungsergebnisse in das Ausgangsregister. Der nach dem letzten CLK-P-Impuls im Ausgangsregister stehende Wert wird als Endergebnis an den D/A-Umsetzer ausgegeben. Dies

erfolgt durch Aktivierung der $\overline{TSM}$-Leitung (Kanal A5). Die Filterverarbeitung – vom Freigabeimpuls bis zum letzten CLK-P – benötigt 6,1 µs bei einer Filtertiefe von M=15. Wie das Zeitdiagramm zeigt, kann die Verarbeitungsdauer (Kanäle A3 und A4) bis zur nächsten Übernahme der Bussteuerung durch den DMA-Prozessor (Kanal A0) genutzt werden. Bei entsprechender Erweiterung der Speicher ist somit – unter Beibehaltung der Zykluszeit von 375 ns für CLK X, Y und CLKP – eine Verarbeitungstiefe von bis zu 40 Koeffizienten möglich.

6 Zusammenfassung

Der vorgestellte Aufbau eines digitalen Signalprozessors bietet eine Reihe von Vorteilen gegenüber herkömmlichen Konzepten. Mit der Möglichkeit, sämtliche am digitalen Verarbeitungsprozeß beteiligten Baugruppen in ein Z80-µP-System einzubeziehen, wird ein hohes Maß an Flexibilität erreicht. Ein Anwender kann damit für die Steuerung des Prozessors auf bekannte Programmiertechniken zurückgreifen. Eine Änderung der Filterstruktur läßt sich durch Austauschen der Koeffizienten und des Mikroprogramms sowie durch Variation der Filtertiefe auf einfache Weise realisieren. Der Platzbedarf ist mit insgesamt vier zusätzlichen Europakarten gering.

Ein Ausbau des bestehenden Systems bereitet keine größeren Schwierigkeiten. Zur Verbesserung der Filterwirkung ist eine Erweiterung der Filtertiefe von 15 auf 90 bei gleichzeitiger Verringerung der Zykluszeit bereits in Vorbereitung, womit auch die Verarbeitungskapazität des Multiplizierer-Akkumulators besser ausgenutzt wird. Außerdem ist eine Ergänzung für den Aufbau eines rekursiven Filters geplant.

Für ein Digitalverarbeitungssystem dieser Art ergeben sich neben der Anwendung im Niederfrequenzbereich eine Reihe anderer Verwendungsmöglichkeiten. So ist beispielsweise der Einsatz als µP-unterstützende Arithmetikeinheit, aber auch die Anwendung in der Meßwerterfassung sowie der Steuerungs- und Regelungstechnik denkbar.

Cand.-Ing. Reinhold Ludwig wurde in Ravensburg geboren und ist in Lockweiler/Saarland aufgewachsen. Seit Anfang 1977 studiert er an der Universität Gesamthochschule Wuppertal Elektrotechnik mit Schwerpunkt Automatisierungstechnik. Gegenstand seiner Studien- und laufenden Diplomarbeit ist die Echtzeitverarbeitung von Niederfrequenzsignalen.
Hobbys: Jagen, Skifahren und Wandern





Dipl.-Ing. Wolfgang Wirwahn ist gebürtiger Remscheider. Er studierte an der TH Aachen Elektrotechnik (Fachrichtung Nachrichtentechnik). Seit 1978 ist er als wissenschaftlicher Mitarbeiter an der Universität Gesamthochschule Wuppertal tätig. Hier beschäftigt er sich mit der digitalen Verarbeitung von Niederfrequenzsignalen.
Hobbys: Familie (1 Tochter), Musik

Der vorgestellte Signalprozessor entstand im Rahmen einer Studienarbeit im Labor Digital- und Impulstechnik der Universität Gesamthochschule Wuppertal unter Leitung von Herrn *Prof. H. Kieseyer.*

Literatur

[1] Datenblätter A/D/A/M-724, MP 1926A, MP 201A. Fa. Analogic, 1980.
[2] Microcomputer Components Data Book. Fa. Mostek, 1981.
[3] Datenblatt TDC 1010J. Fa. TRW, 1979.
[4] *Hug, H.*: Vollintegrierte digitale Multiplizierer für den Einsatz in Mikrocomputersystemen. ELEKTRONIK 1980, H. 1, S. 67...71.

Dr.-Ing. Jürgen Schloß, Dipl.-Ing. Herbert Müller

Schneller Vektorprozessor zum Anschluß an 16-Bit-Mikrocomputer

Herkömmliche Mikrocomputer sind in den meisten Fällen für Echtzeitanwendungen zu langsam. Um ihren Einsatz auf diesem Gebiet dennoch zu ermöglichen, wurde ein Vektorprozessor entwickelt, der als schnelles Zusatzrechenwerk verwendet werden kann. Der Hardwareaufwand liegt bei drei Doppeleuropakarten und ist damit weit geringer als bei den bekannten Array-Prozessoren, die zum Anschluß an Minicomputer und Großrechner konzipiert wurden. Bevorzugte Anwendungsgebiete sind die digitale Signalverarbeitung, die Mustererkennung sowie die Sprach- und Bildverarbeitung. Auch bei Robotern und Bahnsteuerungen fallen in sehr kurzer Zeit viele Rechenoperationen an, die schnell verarbeitet werden müssen. Diese Anforderungen lassen sich mit dem beschriebenen Vektorprozessor erfüllen. Zunächst wird ein Einblick in die Prinzipien der Vektorverarbeitung gegeben, es folgt eine Erläuterung zum Hardwareaufbau.

Durch die Weiterentwicklung der Theorie diskreter Systeme und ihre Anwendung für die digitale Signalverarbeitung [1, 2, 3] in den letzten Jahren hat das Interesse an schnellen Rechenwerken ständig zugenommen. Die Algorithmen der digitalen Signalverarbeitung sind meist außerordentlich rechenintensiv (z. B. in der Bildverarbeitung) und darüber hinaus häufig auch noch sehr zeitkritisch, wenn es um Echtzeitanwendungen geht (z. B. digitale Regler, Filter, Sprachverarbeitung).

Es ist sicherlich möglich, durch den Einsatz von Multiprozessorsystemen den Engpaß in der Rechenleistung zu mildern, doch belegt folgende Überlegung, wie problematisch diese Vorgehensweise ist: Der MOS-Mikroprozessor 8085 kann 8 Bit breite Datenworte verarbeiten. Zur Ausführung einer 16×16-Bit-Multiplikation ist deshalb eine zeitaufwendige Doppelwortarithmetik notwendig, so daß die Multiplikationszeit bei 400 µs liegt. Der Einsatz von zum Beispiel 100 Rechnern würde eine mittlere Multiplikationszeit von 4 µs ergeben und damit aber immer noch um den Faktor 40 über der von einem schnellen Parallelmultiplizierer-IC benötigten Rechenzeit liegen. Probleme des Datentransportes sind hierbei nicht berücksichtigt.

Diese Verhältnisse können durch den Einsatz von MOS-Mikroprozessoren, die 16 Bit verarbeiten und Maschinenbefehle für die Multiplikation zur Verfügung stellen, zwar verbessert werden, doch liegt auch bei diesen Rechnern die Multiplikationszeit für Zweier-Komplement-Zahlen zum Teil noch erheblich über 10 µs, so daß auch hier eine sehr hohe Zahl von Rechnern nötig ist, um die Leistungsfähigkeit eines bipolaren Parallelmultiplizierers zu erreichen. Dabei sind die Steuerungsprobleme beim Einsatz so vieler Prozessoren noch gar nicht berücksichtigt!

Geht man von der Anwendung eines Prozessorsystems für die spezielle Klasse der Algorithmen für die digitale Signalverarbeitung aus, so kann man eine gewisse Homogenität (gleichartige Operationen auf viele Daten) der notwendigen Rechnungen beobachten. Dies führt zu speziellen Rechenwerken, bei denen die Architektur an die Algorithmen angepaßt ist.

Für die Aufgaben der digitalen Signalverarbeitung zeigt sich, daß die Verarbeitung von Vektoren eine immer wiederkehrende Operation ist. Dies wird nachfolgend anhand einiger Beispiele beschrieben.

1 Vektorielle Signaldarstellung

Unter einem Vektor wird hier wie üblich eine geordnete Menge von Zahlen verstanden. Zum Beispiel liefert ein periodisch abtastender Analog-Digital-Umsetzer einen Signalvektor (*Bild 1*), der zum Beispiel unendlich lang sein kann

$\underline{u} = [u_0, u_1, u_2, \ldots, u_n, \ldots]^T$ *)

oder aus Einzelvektoren besteht

$\underline{u} = [\underline{u}_1^T, \underline{u}_2^T, \ldots, \underline{u}_n^T, \ldots]^T$

mit

$$\underline{u}_1 = [u_0, u_1, \ldots u_n] \qquad (1)$$

$$\underline{u}_2 = [u_{n+1}, u_{n+2}, \ldots u_m] \text{ usw.}$$

*) Zur Schreibweise:
Es ist $(u_1, u_2, \ldots) = \begin{bmatrix} u_1 \\ u_2 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}^T$ bzw. $\begin{bmatrix} u_1 \\ u_2 \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} = (u_1, u_2, \ldots)^T$

oder aber aus einzelnen, sich in ihren Elementen überlappenden Vektoren aufgebaut sind:

$\underline{u} = [\underline{u}_1^T, \underline{u}_2^T, \dots \underline{u}_n^T, \dots]^T$

mit zum Beispiel

$$\underline{u}_1 = [u_0, u_1, u_2, u_3]^T \quad (2)$$

$\underline{u}_2 = [u_1, u_2, u_3, u_4]^T$

$\underline{u}_3 = [u_2, u_3, u_4, u_5]^T$ usw.

Die Aufteilung nach Gl. (1) ist naheliegend für ein Fernsehbild, die u_ν sind dann die Bildpunkte, die $\underline{u}_\lambda$ die Zeilen.

Die Darstellung nach Gl. (2) ist vorteilhaft für die digitale Berechnung von Filterfunktionen oder Korrelationen. Da Signalabtastwerte meist gleichberechtigt sind, läßt sich schon erahnen, daß die digitale Verarbeitung von Signalen häufig durch gleichartige Operationen über sehr viele Abtastwerte (u_ν) realisiert wird [4, 5].

2 Erste Anwendung: Eindimensionale Signale

Eine typische Anwendung der digitalen Signalverarbeitung ist die Hoch- bzw. Tiefpaßfilterung von Signalen. Man kann damit beispielsweise eine Gleichspannungskomponente oder Drift entfernen bzw. eine Rauschbefreiung vornehmen. Als Beispiel sei hier ein nichtrekursives digitales Filter 3. Grades angeführt [1]; *Bild 2* zeigt seine Struktur.

Die u_ν seien die Abtastwerte entsprechend der Darstellung in Bild 1, v_ν sind die digital berechneten Ausgangswerte.

Der ν-te Abtastwert des Ausgangssignals wird berechnet durch

$v_\nu = h_0 \cdot u_\nu + h_1 \cdot u_{\nu-1} + h_2 \cdot u_{\nu-2} + h_3 \cdot u_{\nu-3}$

und weiter

$v_{\nu+1} = h_0 \cdot u_{\nu+1} + h_1 u_\nu + h_2 u_{\nu-1} + h_3 u_{\nu-2}$ usw.

oder in vektorieller Schreibweise:

$$v_\nu = (h_0, h_1, h_2, h_3) \begin{bmatrix} u_\nu \\ u_{\nu-1} \\ u_{\nu-2} \\ u_{\nu-3} \end{bmatrix} = \underline{h}\,\underline{u}^T$$

$$\text{bzw. } v_\nu = \sum_{\lambda=0}^{3} h_\lambda \cdot u_{\nu-\lambda} \quad (3)$$

Ein Signalprozessor, der als nichtrekursives digitales Filter programmiert ist, muß also zur Bestimmung eines Filterausgangswertes das „innere Produkt" zweier Vektoren berechnen. Seine Architektur sollte die Berechnung von Ausdrücken wie in Gl. (3), also das Bilden von Summen von Produkten, vorteilhaft ermöglichen.

Um einen ersten Eindruck von den notwendigen Rechenleistungen eines Signalprozessors zu geben, soll überschlägig die Rechenzeit zur Ermittlung eines Ausgangssignalwertes abgeschätzt werden. Das Filter nach Bild 2 benötigt für einen Wert v_ν:

4 Multiplikationen,
3 Additionen und mindestens
3 Speichertransfers.

Ausgehend von einem Mikroprozessor 8085 sind hierzu 1...2 ms notwendig. Das heißt, die Abtastrate kann 0,5...1 kHz sein, was entsprechend dem Shannonschen Abtasttheorem einer Signalbandbreite von etwa 0,25...0,5 kHz entspricht. Diese Bandbreite ist weit geringer, als zum Beispiel für Telefonqualität gefordert wird. Möchte man ein Audio-Signal in HiFi-Qualität (Bandbreite 20 kHz) verarbeiten, muß der Signalprozessor bei diesem Beispiel bereits eine um den Faktor 20...40 höhere Rechenleistung bieten.

Ohne detaillierte Begründung sei angemerkt, daß sich auch für Algorithmen wie die schnelle Fouriertransformation (FFT) oder auch für Korrelationen vektorielle Algorithmenbeschreibungen angeben lassen [4].

Der gezeigte Filteralgorithmus ist auch auf Realisierungen digitaler Regler und digitaler Interpolatoren, z. B. für Bahnkurvensteuerungen (Industrieroboter), anwendbar.

3 Zweite Anwendung: Zweidimensionale Signale

Die Anwendung von Vektoroperationen auf zweidimensionale Signale soll anhand der Bildverarbeitung gezeigt werden. Eine typische Aufgabe ist hierbei die Kantendetektion. Sie kann zum Beispiel mit einem „Laplace-Operator" durchgeführt werden, der ein digitales zweidimensionales Hochpaßfilter darstellt [6, 7].

Nachfolgend wird gezeigt, wie die Filteroperation vor sich geht.

Es sei

$$\begin{matrix} a_{11} & a_{12} & a_{13} & a_{14} & \dots \\ a_{21} & a_{22} & a_{23} & a_{24} & \dots \\ a_{31} & a_{32} & a_{33} & a_{34} & \dots \\ a_{41} & a_{42} & a_{43} & a_{44} & \dots \\ \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \end{matrix}$$

das zu bearbeitende Bild mit den Bildpunkten a_{xy} und

$$\begin{matrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{matrix}$$

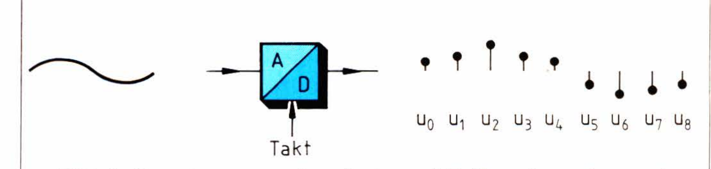


Bild 1. Das Ausgangssignal eines A/D-Umsetzers kann als Vektor beschrieben werden

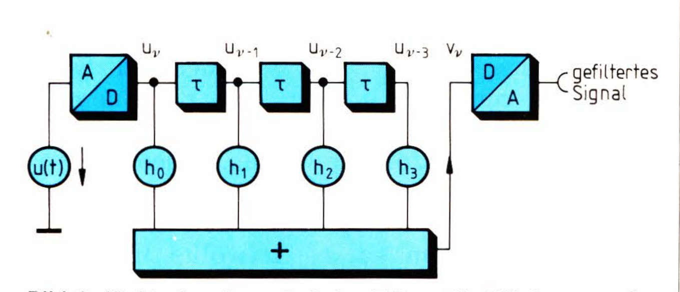


Bild 2. Nichtrekursives digitales Filter: Die Blöcke τ verzögern die Abtastwerte um einen Takt

der Filteroperator, wobei h_{xy} die Filterkoeffizienten sind (für Laplace-Operator:
$h_{11} = h_{13} = h_{31} = h_{33} = 0$,
$h_{12} = h_{21} = h_{23} = h_{32} = -1$,
$h_{22} = +4$),
dann ist das gefilterte Bild

$$\begin{matrix} b_{11} & b_{12} & b_{13} & b_{14} & \dots \\ b_{21} & b_{22} & b_{23} & b_{24} & \dots \\ b_{31} & b_{32} & b_{33} & b_{34} & \dots \\ b_{41} & b_{42} & b_{43} & b_{44} & \dots \\ \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \\ \cdot & \cdot & \cdot & \cdot & \end{matrix}$$

wobei die Bildpunkte b_{xy} durch z. B.

$$b_{22} = a_{11} \cdot h_{11} + a_{12} \cdot h_{12} + a_{13} \cdot h_{13} + a_{21} \cdot h_{21} + a_{22} \cdot h_{22} + a_{23} \cdot h_{23} + a_{31} \cdot h_{31} + a_{32} \cdot h_{32} + a_{33} \cdot h_{33} \text{ usw.}$$

zu berechnen sind.

Für jede Zeile des Ausgangsbildes läßt sich dann folgende vektorielle Darstellung angeben:

$$\begin{bmatrix} b_{22} \\ b_{23} \\ b_{24} \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} = \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} h_{11} + \begin{bmatrix} a_{12} \\ a_{13} \\ a_{14} \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} h_{12} + \dots + \begin{bmatrix} a_{33} \\ a_{34} \\ a_{35} \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} h_{33} \quad (4)$$

oder: $\underline{b} = \underline{a_1}\, h_{11} + \underline{a_2}\, h_{12} + \dots + \underline{a_9}\, h_{33}$

Aus Gleichung (4) geht hervor, daß der ausführende Signalprozessor neun Vektoren $\underline{a_\nu}$ mit Filterkoeffizienten h_{xy} multiplizieren und die so erhaltenen Zwischenergebnisvektoren aufakkumulieren muß.

Dies läßt sich darstellen als

$\underline{b}: = \underline{a_1}\, h_{11}$; $\underline{b}: = \underline{a_2}\, h_{12} + \underline{b}$; ... $\underline{b}: = \underline{a_9}\, h_{33} + \underline{b}$

das heißt, durch neunmaligen Aufruf eines Vektorbefehls der Form

Vektor A = Vektor B · Konst + Vektor C (5).

In dem später beschriebenen Vektorprozessor ist ein solcher Befehl implementiert, er wird mnemonisch beschrieben durch

VMADD ≙ Vektor Multiply ADD ≙ $\underline{A} \cdot \underline{B} + \underline{C} \rightarrow \underline{D}$.

Die ausführliche Beschreibung dieses Beispieles ist als kurze Einführung in die Programmierverfahren für Vektorprozessoren gedacht. Es sei lediglich erwähnt, daß sich auch für nichtlineare Operationen, für die 2D-FFT bzw. 2D-Korrelation, vektorielle Darstellungen der Algorithmen angeben lassen.

4 Dritte Anwendung: Mustererkennung

In der Mustererkennung unterscheidet man zwischen Algorithmen zur Merkmalsextraktion, die einen Merkmalsvektor liefern, und solchen zur Klassifikation, die diesen Merkmalsvektor verarbeiten. Insbesondere im Bereich der Merkmalsextraktion wird sehr häufig eher intuitiv vorgegangen, und es würde den Rahmen dieses Beitrages bei weitem sprengen, wollte man hierauf detailliert eingehen [8].

Zur Gewinnung von Merkmalen bei der Spracherkennung wird häufig eine Spektralanalyse durchgeführt und die Leistung in bestimmten Frequenzbändern als Merkmal benutzt. Diese Analyse ist durch Vektoroperationen beschreibbar, wie bereits in Abschnitt 2 erwähnt wurde.

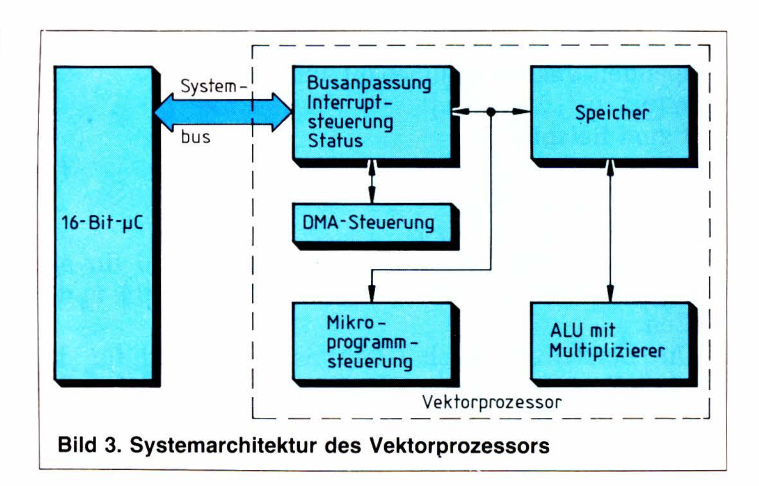


Bild 3. Systemarchitektur des Vektorprozessors

In der Bildverarbeitung besteht häufig die Aufgabe, Objekte zu erkennen (z. B. Positionieren von Robotern), zu zählen (z. B. medizinische Anwendung: „Zahl der Blutkörper") oder zu analysieren (z. B. Werkstoffprüfung).

Bei all diesen Anwendungen müssen Strukturen bzw. Formen erkannt und mit Merkmalen (also Zahlen) beschrieben werden. Eine typische Aufgabe hierbei ist die Bestimmung der Schwerpunktkoordinaten eines Objektes im Bild, dessen Umrißlinien durch Verfahren der Kantendetektion (siehe Abschnitt 3) ermittelt wurden.

Um die Leistungsfähigkeit der vektoriellen Algorithmen auch für diese Aufgabenbereiche zu demonstrieren, seien folgende Angaben gemacht: Ein Bild mit 512 × 512 Bildpunkten habe 40 Objekte (≙ geschlossene Konturen). Um alle Objekte zu finden, gibt es einen Algorithmus, der im Mittel die Aufgabe mit 32 Mio. Vektorvergleichsoperationen löst. Hierzu benötigt ein Prozeßrechner PDP-11/34 etwa 12,4 min, der beschriebene Vektorprozessor nur 30 s, obwohl sein Hardwareaufwand bei nur drei Doppeleuropakarten liegt.

5 Anforderungen an einen Vektorprozessor

Aus den vorangegangenen Beispielen geht hervor, daß eine zentrale Operation für die beschriebenen Beispiele die multiplikative und additive Verknüpfung von Elementen von Vektoren darstellt. In der Auswahl der richtigen Vektorbefehle und deren Implementierung in einem Vektorprozessor liegt das eigentliche System-Know-how. Es sind nämlich neben den klassischen Algorithmen der digitalen Signalverarbeitung wie Filterung oder FFT meistens noch weitere Operationen notwendig. Insbesondere für die Mustererkennung werden häufig nichtlineare Verfahren eingesetzt, die im einfachsten Fall aus Schwellwertoperationen, Maximum- bzw. Minimumsuche oder Datensortierroutinen bestehen.

Auch diese Rechenoperationen laufen über große Datenmengen – also Vektoren – ab.

Der nachfolgend beschriebene Prozessor ist deshalb mikroprogrammierbar [10]. Dadurch werden hauptsächlich zwei Vorteile erreicht:

- Der Prozessor kann an neue Aufgaben jederzeit optimal angepaßt werden.
- Dem Benutzer kann ein Basisbefehlssatz zur Verfügung gestellt werden, der ihn durch seine Komplexität (z. B. vollständige FFT durch Aufruf nur eines Maschinenbefehls) von der mühsamen und zeitaufwendigen Aufgabe der Algorithmenprogrammierung weitgehend befreit.

6 Systemarchitektur

Bild 3 zeigt die Systemarchitektur des Vektorprozessors. Er wird an den 16-Bit-Bus eines Mikrocomputers angeschlossen und nach dem Prinzip des Memory-Mapping angesteuert. Er hat einen eigenen schnellen Speicher in Schottky-Technik, der als Zweitorspeicher aufgebaut und somit für den Benutzer völlig transparent ist. Die zu bearbeitenden Vektoren werden von angeschlossenen Mikrocomputern in diesen Speicher geladen oder von einer im Vektorprozessor vorhandenen DMA-Steuerung automatisch mit bis zu 4 MByte/s von einer am Bus angeschlossenen Datenquelle geholt. Dieses Verfahren gewährleistet eine hinreichend schnelle Datenübertragung.

Eine Operation im Vektorprozessor wird ausgelöst durch die Übertragung des Befehlscodes vom µC in eine bestimmte Speicherzelle des Vektorprozessors. Eine Besonderheit bietet der Vektorprozessor für den Fall, daß eine Vektorbefehlsfolge ständig wiederholend aufgerufen werden soll: Es ist möglich, in seinem Speicher eine fast beliebige Zahl von Befehlen abzulegen, die sich die Steuerung des Vektorprozessors dann selbständig mit optimaler Geschwindigkeit abholt und bearbeitet.

Die Synchronisation des Vektorprozessors mit dem µC kann über Interrupt oder die Abfrage eines Statusbits („Vektorprozessor Busy") vorgenommen werden. Außerdem lassen sich über das Statuswort Fehlercodes an das µC-System übermitteln. Beide Prozessoren sind autonom; während der Vektorprozessor arbeitet, kann der µC eigene Programme ausführen.

7 Hardware

Einen detaillierten Einblick in die Hardwarearchitektur gibt *Bild 4*. Die wesentlichen Komponenten sind

- Mikroprogrammsteuerung (Sequencer Typ Am 2909)
- Rechenwerk
- 4 Speicher mit 4 Adreßrechnern und 2-Tor-Steuerung
- DMA-Steuerwerk

Die Architektur wurde entworfen mit dem Ziel, die bereits erwähnte Operation

$$\underline{A} = \underline{B} \cdot \underline{C} + \underline{D} \qquad (6)$$

so schnell wie es mit der derzeitigen TTL-Technik möglich ist, zu berechnen. Als zentrale Rechenkomponenten wurden deshalb ein schneller 16-Bit-Parallelmultiplizierer (MPY-16 HJ; Produkt-Bildung in 150 ns) sowie eine ALU (74 F 381) ausgewählt [9].

Eine bemerkenswerte Besonderheit der Architektur ist, daß jedem Speicher ein eigener Adreßrechner zugeordnet ist. Dadurch gewinnt das System eine außerordentlich hohe Flexibilität bei Speicherzugriffen. Es wird

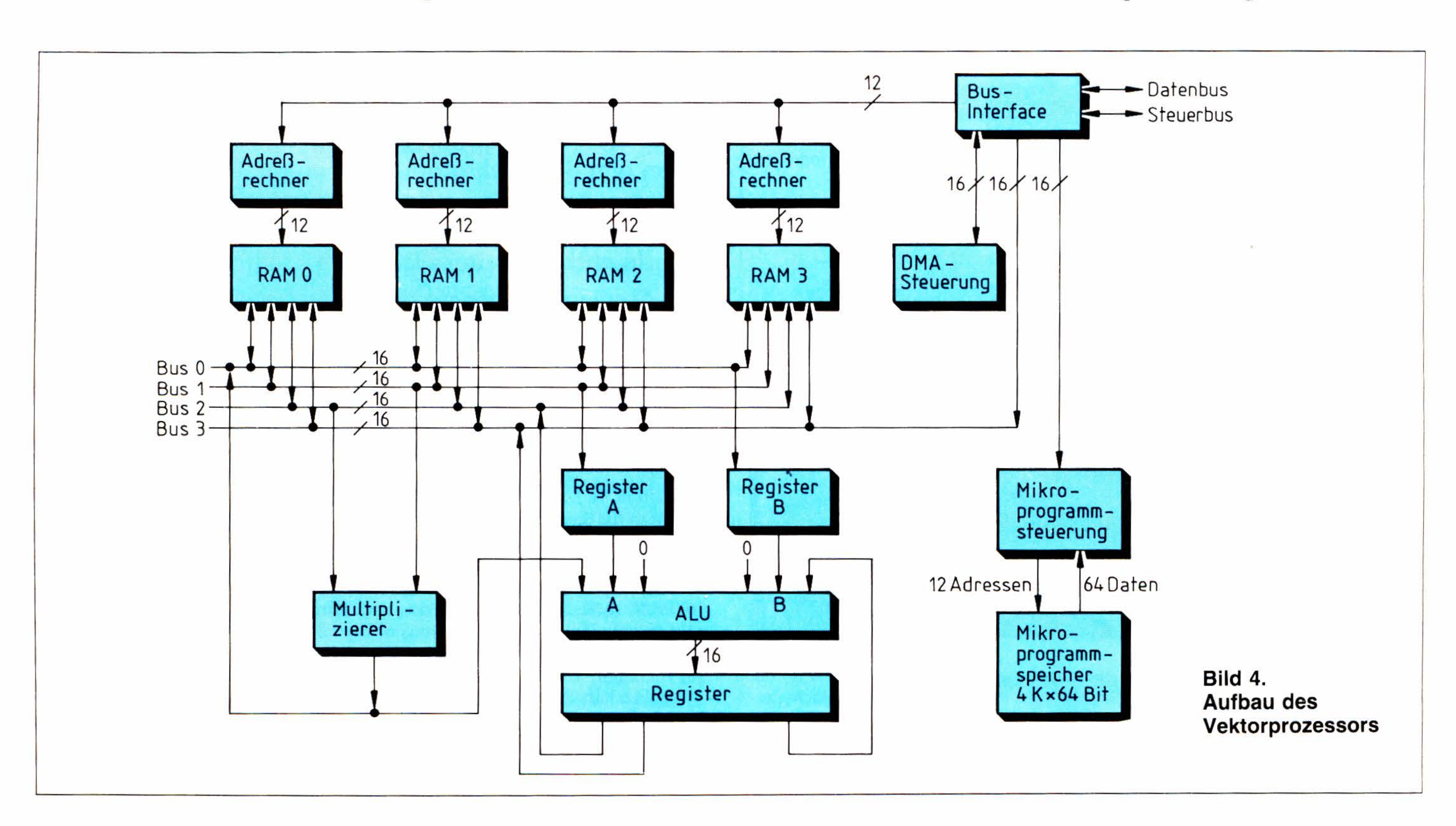


Bild 4. Aufbau des Vektorprozessors

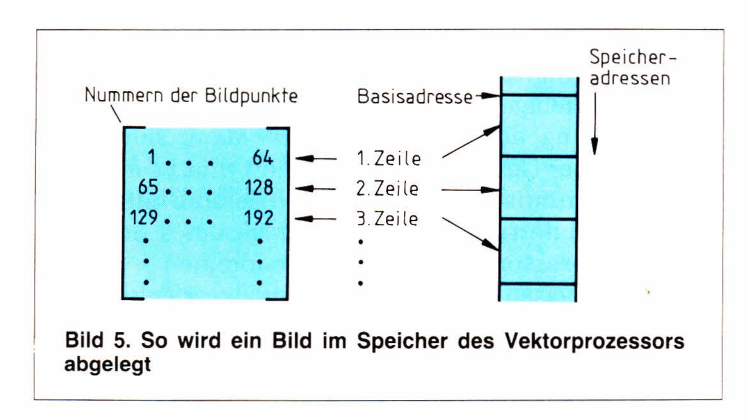
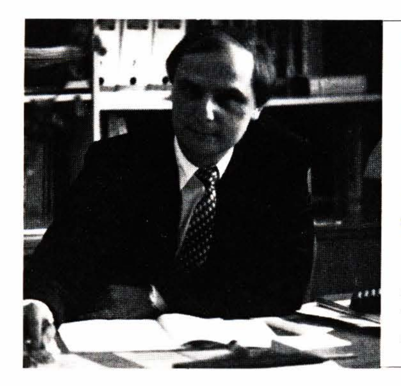


Bild 5. So wird ein Bild im Speicher des Vektorprozessors abgelegt

dadurch möglich, einen Vektor durch drei Parameter zu beschreiben:
- Anfangsadresse (z. B.: = Basisadresse)
- Zahl der Elemente (z. B.: = 64)
- Adreßoffset (z. B.: = 64)

Die Bedeutung des Adreßoffsets soll an einem Beispiel erklärt werden: Im Speicher des Prozessors sei ein Bild abgelegt, wobei die Zeilen aufeinanderfolgend mit aufsteigenden Adressen abgelegt sind (*Bild* 5). Wenn nun eine Bildspalte bearbeitet (und somit adressiert) werden soll, so sind die Adressen z. B. für die zweite Spalte 2, 66, 130, ... usw.

Diese außerordentlich einfach zu spezifizierenden Parameter müssen vom Programmierer mit dem Befehlscode an den Vektorprozessor übergeben werden, die



Dr.-Ing. Jürgen Schloß studierte an der Universität Erlangen Nachrichtentechnik/Elektronik. Er war dann am Institut für Nachrichtentechnik wissenschaftlicher Mitarbeiter, später Assistent. Er promovierte über die Beziehung von Rechnerarchitekturen zu Algorithmen der digitalen Signalverarbeitung im Hinblick auf die Realisierung von Hochgeschwindigkeitsrechnern. Seit 1980 leitet er bei der Firma Diehl, Nürnberg, eine Entwicklungsabteilung für Mikroprozessoren/Sensoren.



Dipl.-Ing. (TU) Herbert Müller wurde in Schwabach geboren. Er studierte an der Universität Erlangen Elektrotechnik, Fachrichtung Nachrichtentechnik. Nach vierjähriger Tätigkeit am Zentralinstitut für Biomedizinische Technik ist er seit 1981 bei der Firma Diehl, Nürnberg, in der Entwicklungsabteilung von Mikrocomputersystemen tätig.
Hobbys: Elektronik-Selbstbau, Hausbau.

Technische Daten des Vektorprozessors

Technologie	Schottky-TTL, FAST-TTL
Zykluszeit	150 ns
Rechenleistung	max. 13 Mio. Operationen/s
Speicherbandbreite	ca. 50 MByte/s
Steuerung	64 Bit horizontal mikroprogrammiert maximal 4 KWorte über „Piggy-Back"-Platine erweiterbar
Interface	für 16-Bit-μCs wie Z8000, 68000, 8086
Speicher	4 Speichermodule jeweils 4 K x 16 Bit + 4 Adreßrechner 2-Tor-Steuerung
Rechenzeiten	nichtrekursives digitales Filter 10. Grades: 4 μs/Abtastwert FFT; 1024 komplexe Punkte: ca. 17 ms 1024 reelle Punkte: ca. 8 ms
Aufbau	3 Doppeleuropakarten ca. 250 Bausteine
Leistungsverbrauch	ca. 40 W

Hardware führt dann automatisch sämtliche Adreßrechnungen durch. Die Adreßrechner sind mit geringstem Hardwareaufwand durch LSI-Bausteine des Typs Am 2932 realisiert.

Bei vorgegebenen Leistungsdaten war es vorrangiges Entwicklungsziel, durch Verwendung modernster Bauelemente den Hardwareaufwand so gering wie möglich zu halten. Es wurden Bausteine der Serie Am 2900 sowie der schon erwähnte Multiplizierer MPY-16 HJ verwendet. Die notwendige SSI-Logik wurde, soweit möglich, mit PALs realisiert. Aus Geschwindigkeitsgründen wurden vielfach FAST-Bausteine der Serie 74 F XX eingesetzt.

Neben der optimierten Architektur sorgt die „hardwarenahe" Mikroprogrammierung für die außerordentlich hohe Rechenleistung des vorgestellten Konzeptes. Es wird ein 64 Bit breites horizontales Mikrowort verwendet, dabei sind vollständige Algorithmen wie FFT, Korrelation oder digitale Filter als Mikroprogramm neben den Basisvektoroperationen realisierbar. Dadurch wird die Anwendung des Prozessors stark vereinfacht und gleichzeitig seine Leistungsfähigkeit optimiert. Die *Tabelle* enthält weitere Details.

Literatur

[1] *Schüßler, H. W.*: Digitale Systeme zur Signalverarbeitung. Springer, 1973.
[2] *Karphus, W. J., Cohen, D.*: Architectural and Software Issues in the Design and Application of Peripheral Array Processors. IEEE Computer, Sept. 1981.
[3] *Alexander, P.*: Array Processor Design Concepts. Computer Design, Dez. 1981.
[4] *Schloß, H. J.*: Zum Entwurf und zur Anwendung programmierbarer Prozessoren für die digitale Signalverarbeitung. Ausgewählte Arbeiten über Nachrichtensysteme, Nr. 40, Erlangen, 1980.
[5] *Kolb, H. J.*: μP in der digitalen Signalverarbeitung. Mikroprozessoren und ihre Anwendungen. Oldenburg, 1979.
[6] *Pratt, W. K.*: Digital Image Processing. John Wiley & Sons, New York, 1979.
[7] *Oppenheim, A. V.*: Applications of Digital Signal Processing. Prentice Hall, 1978.
[8] *Niemann, H.*: Methoden der Mustererkennung. Akademische Verlagsgemeinschaft Frankfurt a. M., 1974.
[9] *Herg, H.*: Vollintegrierte digitale Multiplizierer für den Einsatz in Mikrocomputern. ELEKTRONIK 1980, H. 1, S. 67...71.
[10] *Schloß, H. J., Müller, H.*: Ein mikroprogrammierbares Entwicklungssystem für bipolare Mikroprozessorelemente. ELEKTRONIK 1978, H. 8, S. 40...47.

Dipl.-Ing. Gerd Ballenberger, Dipl.-Ing. Ernst Zauper

„Nanosignalprozessor" – Alternative zu integrierten Signalprozessoren

Die digitale Verarbeitung von Signalen läßt sich mit unterschiedlichen Geräten ausführen, die zumindest zur Zeit noch gewisse Nachteile haben: Großrechner sind für den ständigen Einsatz zu teuer, schnelle spezielle Prozessoren sehr aufwendig (z. B. [1]), Mikroprozessoren sind für viele Aufgaben zu langsam, integrierte Signalprozessoren haben (noch) zu kleine Speicher, spezielle festverdrahtete Hardware bietet nicht die wünschenswerte Flexibilität.

Hier wird der „Nanosignalprozessor NSP 81" vorgestellt, der in einer Diplomarbeit am Lehrstuhl für Nachrichtentechnik der Universität Erlangen-Nürnberg entstand. Es wird gezeigt, daß mit verhältnismäßig geringem Aufwand (209 Bausteine) die Realisierung eines zwar spezialisierten, aber programmierbaren und daher flexiblen, vor allem schnellen Signalprozessors möglich ist, der die erwähnten Nachteile weitgehend vermeidet.

1 Konzept und Randbedingungen

Bei der Entwicklung des Prozessors waren folgende Gesichtspunkte zu berücksichtigen: Der Prozessor muß speziell für die schnelle Ausführung der in der digitalen Signalverarbeitung gebräuchlichen Algorithmen (s. Abschnitt 2) geeignet sein und dabei eine möglichst hohe Flexibilität und damit Programmierbarkeit besitzen. Dies ist gleichbedeutend mit der Forderung nach hardwarenaher Mikroprogrammierung, dem Vorhandensein eines von außen zugänglichen, d. h. ladbaren Mikroprogrammspeichers, und, wie noch gezeigt wird, einer flexiblen Datenfluß-Struktur.

Die Möglichkeit zur Steuerung und Beobachtung bedeutet, daß bei Bedarf auch innerhalb des Prozessors ablaufende Vorgänge beobachtet werden können. Damit erübrigt sich dann die Software-Simulation des Prozessors, und neuentwickelte Mikroprogramme können unmittelbar auf der Prozessor-Hardware getestet werden. Direkte Folge dieses Wunsches nach Steuer- und Beobachtungsmöglichkeiten ist der Einsatz eines Bedienrechners, der einerseits den Aufruf vorhandener Mikroprogramme ermöglicht und damit verbundene Funktionen wie Parameterabfrage und -umrechnung, Rechenzeitüberwachung usw. ausführt, andererseits aber auch als Hilfsmittel bei der Entwicklung neuer Mikroprogramme dient.

Die dritte entscheidende Forderung stellt einen Widerspruch in sich dar: Es sollte eine möglichst hohe Verarbeitungsgeschwindigkeit bei gleichzeitig mäßigem Bauelementeaufwand erreicht werden.

Die Konsequenzen dieser widersprüchlichen Forderung sind in *Bild 1* dargestellt. Zunächst läßt sich dem Wunsch nach minimalem Aufwand dadurch entgegenkommen, daß als zentrales Arithmetikelement ein integrierter VLSI-Signalverarbeitungsbaustein (Multiplizierer-Akkumulator, „MAC") Verwendung findet, der die

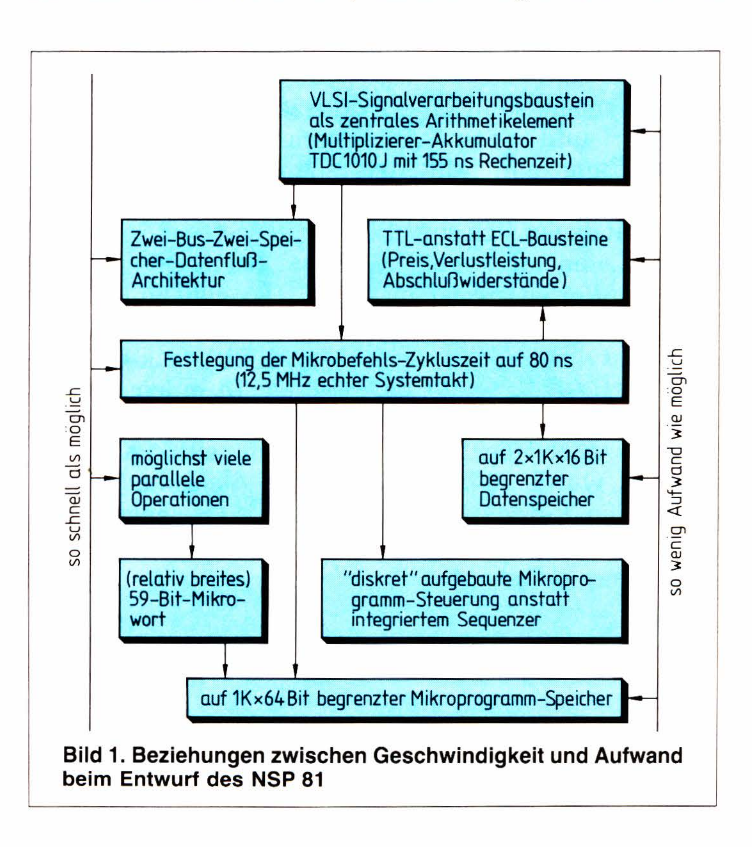


Bild 1. Beziehungen zwischen Geschwindigkeit und Aufwand beim Entwurf des NSP 81

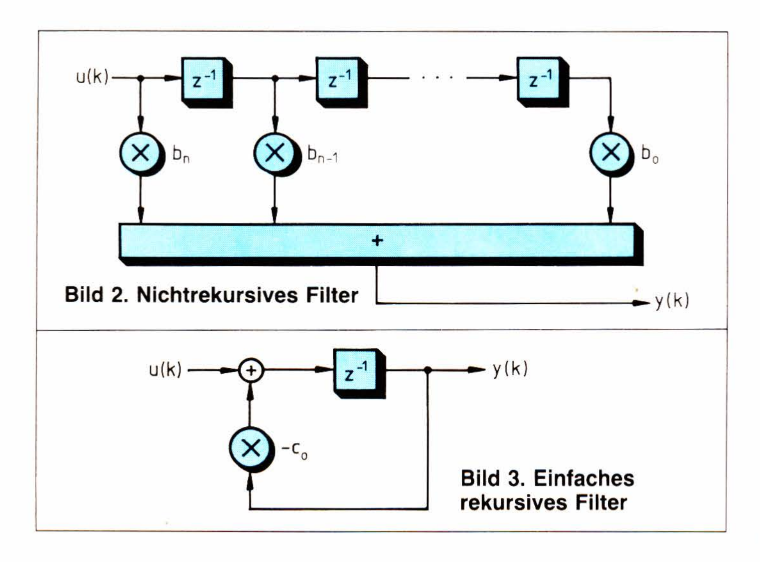


Bild 2. Nichtrekursives Filter

Bild 3. Einfaches rekursives Filter

Funktionen Multiplikation, Addition/Subtraktion und Akkumulation auf einem Chip ausführt. Dieser Baustein läßt sich zweckmäßig (s. Abschnitt 2) in eine Zwei-Bus-Zwei-Speicher-Datenfluß-Architektur eingliedern, die einen hohen Datendurchsatz ermöglicht und damit auch den Geschwindigkeitsanforderungen entgegenkommt: Der MAC benötigt für eine 16×16-Bit-Multiplikation und anschließende Addition 155 ns. Nachdem es sich in einer früheren Arbeit als günstig herausgestellt hatte, während eines solchen „Rechenzyklus" zwei Mikrobefehle auszuführen [2], wurde für den NSP81 die Mikrobefehls-Zykluszeit auf 80 ns festgelegt, d. h. der Prozessor arbeitet mit einem echten 12,5-MHz-Einphasen-Systemtakt. Diese Taktfrequenz erlaubt noch die Verwendung von TTL-Bausteinen, wobei sich gegenüber ECL-Technik die Kosten durch den geringeren Bausteinpreis, das Wegfallen von Abschlußwiderständen und durch die geringere Verlustleistung in Grenzen halten.

Andererseits stellt eine Zykluszeit von 80 ns für die Speicher und deren Adressierung eine erhebliche Anforderung dar. Mit den schnellsten verfügbaren 1-K×4-Bit-NMOS-RAMs standen Bausteine zur Verfügung, mit denen sich sowohl die vorgegebene Zykluszeit als auch eine angestrebte Datenspeichergröße von zweimal 1 K Worten erreichen ließen. Das gleiche Problem tritt nochmals beim Mikroprogrammspeicher auf: für das relativ breite Mikrowort (≥ 59 Bit), das mehrere gleichzeitig ablaufende Operationen in einem Zyklus ermöglicht, werden ziemlich viele Bausteine benötigt, weshalb auch hier die Speichertiefe auf 1 K Worte begrenzt wurde. Weiterhin verbietet die hohe Taktfrequenz den Einsatz eines integrierten Sequenzers. Die Mikroprogrammsteuerung ist deshalb mit MSI-TTL-Bausteinen aufgebaut.

2 Architektur

Die Architektur eines Spezialprozessors wird von den typischen Anwendungen beeinflußt, d. h. durch die Algorithmen, für deren Ausführung der Prozessor besonders geeignet sein soll. Beispiele sind:

a) Nichtrekursive Filterung
Die Struktur in *Bild 2* errechnet aus einer Eingangsfolge u(k) eine Ausgangsfolge

$$y(k) = \sum_{\upsilon=0}^{n} b_\upsilon \cdot u(k - n + \upsilon).$$

Dafür sind die folgenden Operationen nötig:
- Entgegennehmen und Speichern von Signalwerten. Die durch „z^{-1}" gekennzeichneten Blöcke sind durch RAM-Speicher realisierte Verzögerer, die die einlaufenden Signal-Abtastwerte u(k) jeweils für die Dauer einer Abtastperiode speichern.
- Multiplikation von Signalwerten mit Filterkoeffizienten ($b_n...b_0$);
- Addition bzw. Akkumulation der Produkte und Ausgabe des Endergebnisses.

b) Rekursive Filterung
Aus der Blockschaltung des sehr einfachen rekursiven Filters in *Bild 3*, das eine Folge

$$y(k) = u(k - 1) - c_0 \cdot y(k - 1)$$

berechnet, geht bereits ein prinzipieller Unterschied zur nichtrekursiven Filterung hervor: außer den dort erwähnten Operationen besteht hier noch die Notwendigkeit, Rechenergebnisse in den Prozessor zurückzuführen und abzuspeichern.

c) Für die Berechnung der Korrelierten

$$y(k) = \sum_{\kappa=0}^{\infty} u_1(\kappa)u_2(\kappa - k)$$

zweier Wertefolgen $u_1(k)$ und $u_2(k)$ benötigt der Prozessor zwei Eingänge.

Alle in a)...c) dargestellten notwendigen Operationen sind auf der sehr einfachen Datenfluß-Struktur von *Bild 4* realisierbar. Diese Grundstruktur besteht eigentlich nur aus dem Multiplizierer-Akkumulator-Rechen-



Dipl.-Ing. G. Ballenberger (links) studierte an der Universität Erlangen-Nürnberg von 1974 bis 1980 Elektrotechnik. Nach Abschluß des Studiums wurde er wissenschaftlicher Mitarbeiter am dortigen Lehrstuhl für Nachrichtentechnik.
Hobbys: klassische Musik, Surfen, Frankreich

Dipl.-Ing. E. Zauper beendete 1981 an der Universität Erlangen-Nürnberg das Elektrotechnik-Studium. Seit Ende 1981 ist er bei TeKaDe Nürnberg im Bereich Nebenstellentechnik tätig.

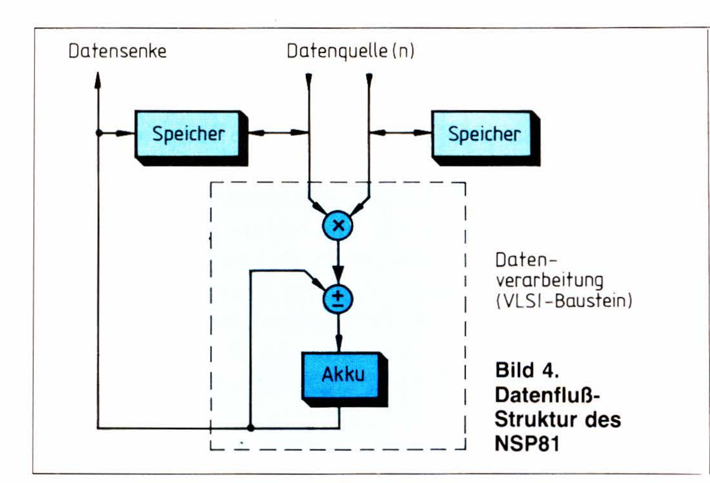


Bild 4. Datenfluß-Struktur des NSP81

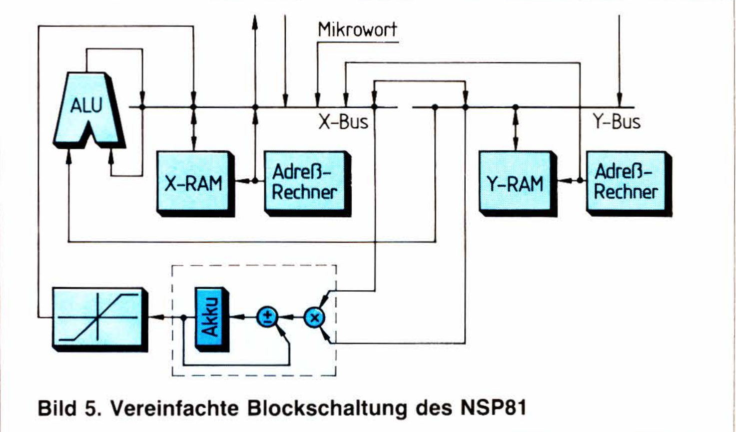


Bild 5. Vereinfachte Blockschaltung des NSP81

baustein, zwei Speichern für die Signalwerte bzw. Filterkoeffizienten sowie zwei Signaleingängen und einem Ausgang, der über den Signalwertspeicher die Rückführung von Ergebnissen in den Prozessor erlaubt.

Die Realisierung einer solchen Minimalstruktur kann für sehr spezielle Aufgaben durchaus interessant sein. Zielt man jedoch auf einen flexibleren Prozessor ab, der die Möglichkeiten der Mikroprogrammierbarkeit weitgehend ausnützt und die möglichst schnelle Ausführung auch anderer Algorithmen erlaubt, so wird man die Minimalstruktur um einige Details erweitern. *Bild 5* zeigt die vereinfachte Gesamt-Blockschaltung des NSP81. Neben den unterlegten Schaltungsteilen, die der Minimalstruktur von Bild 4 entsprechen, sind die folgenden vier Erweiterungen enthalten:

- Durch ein System zweier miteinander gekoppelter und in die Datenflußstruktur eingegliederter Datenbusse besteht die Möglichkeit, eine große Anzahl interner und externer Daten-Quellen und -Senken direkt miteinander zu verbinden, so daß sich jeweils für bestimmte Algorithmen optimale Datenwege ergeben. Mit jedem Mikrowort läßt sich so eine neue Prozessorstruktur programmieren.
- Die Rechenleistung des Prozessors kann erhöht werden, wenn man den Multiplizierer-Akkumulator von Operationen entlastet, für die er nicht besonders geeignet ist (z. B. einfache Addition zweier Operanden, logische Operationen), und dafür eine separate ALU einsetzt. Diese kann z. B. aus Bit-Slice-Elementen aufgebaut sein, oder, wenn alle Operationen innerhalb eines Zyklus ausgeführt werden sollen, aus schnellen, dafür weniger flexiblen TTL-ALU-Bausteinen.
- Die Datenspeicher werden zweckmäßigerweise mit separaten Adreß-Rechnern ausgestattet, die aus wenigen Zähler-, Addierer-, Multiplexer- und Register-Bausteinen aufgebaut werden können, und eine schnelle und flexible Adressierung ermöglichen.
- Für manche Algorithmen muß das Auftreten von arithmetischen Über- bzw. Unterläufen unbedingt verhindert werden [3]. Eine in Hardware realisierte, automatisch arbeitende Begrenzerkennlinie spart Programmieraufwand und Rechenzeit.

Der Mikroprogramm-Speicher (*Bild 6*) ist vom Steuerrechner aus beschreibbar; ebenso können (bei angehaltenem Systemtakt) einzelne Mikrobefehle direkt in das Pipeline-Register geladen, d. h. in ein bestehendes Programm eingeschoben werden. Der Adreß-Stack erlaubt die Ausführung von Unterprogrammen; der Repeater dient als Schleifenzähler. Im Pipeline-Register sind die zur Steuerung der verschiedenen Schaltungsteile benötigten Mikrowort-Felder.

3 Software

Auf dem Steuer-Rechner, einem 8085-System, steht im wesentlichen folgende Software zur Verfügung:
- ein Text-Editor für das Erstellen mnemonisch codierter Mikroprogramme;

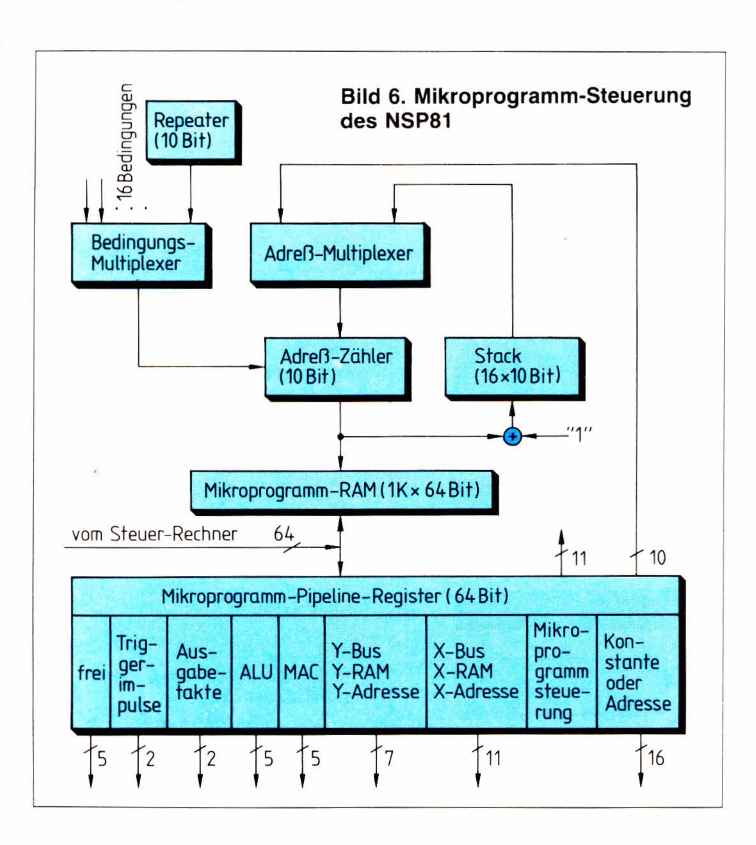


Bild 6. Mikroprogramm-Steuerung des NSP81

– ein Assembler, der für jeden zu übersetzenden Mikrobefehl von einem NOP-Mikrowort ausgeht, und anhand einer Opcode-Tabelle diejenigen Bits in ihren aktiven Zustand versetzt, die zu den Teilbefehlen (Mnemonics) des Mikrobefehls gehören;
– ein Bedien- und Debug-Programm, das Laden, Modifizieren, Starten und Anhalten von Mikroprogrammen erlaubt sowie das Betrachten und Verändern von Register- und Speicherinhalten, Adressen usw.
– ein Bedienprogramm für die bisher programmierten Algorithmen [4] (in Klammern Programmlänge):
  1 nichtrekursives, linearphasiges Filter (12 Worte)
  2 nichtrekursives Filter in direkter Struktur (9 Worte)
  3 rekursives Filter als Kaskade von Blöcken 2. Grades (24 Worte)
  4 rauscharmes Zustandsraum-Filter nach Zeman und Lindgren [5] (50 Worte)
  5 „statische" Korrelation (22 Worte) gleitende Korrelation über kontinuierliche Folgen (51 Worte).

Die mit den digitalen Filtern 1...4 erzielbaren maximalen Abtastfrequenzen sind in *Bild 7* gezeigt.

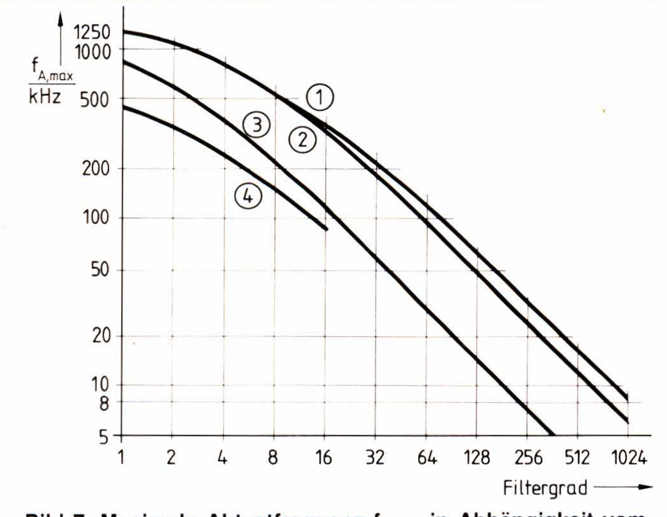


Bild 7. Maximale Abtastfrequenz $f_{A,max}$ in Abhängigkeit vom Filtergrad
① Linearphasiges nichtrekursives Filter
② Nichtlinearphasiges nichtrekursives Filter
③ Rekursives Filter (Kaskadenform, Blöcke 2. Grades)
④ Filteralgorithmus nach *Zeman* und *Lindgren*

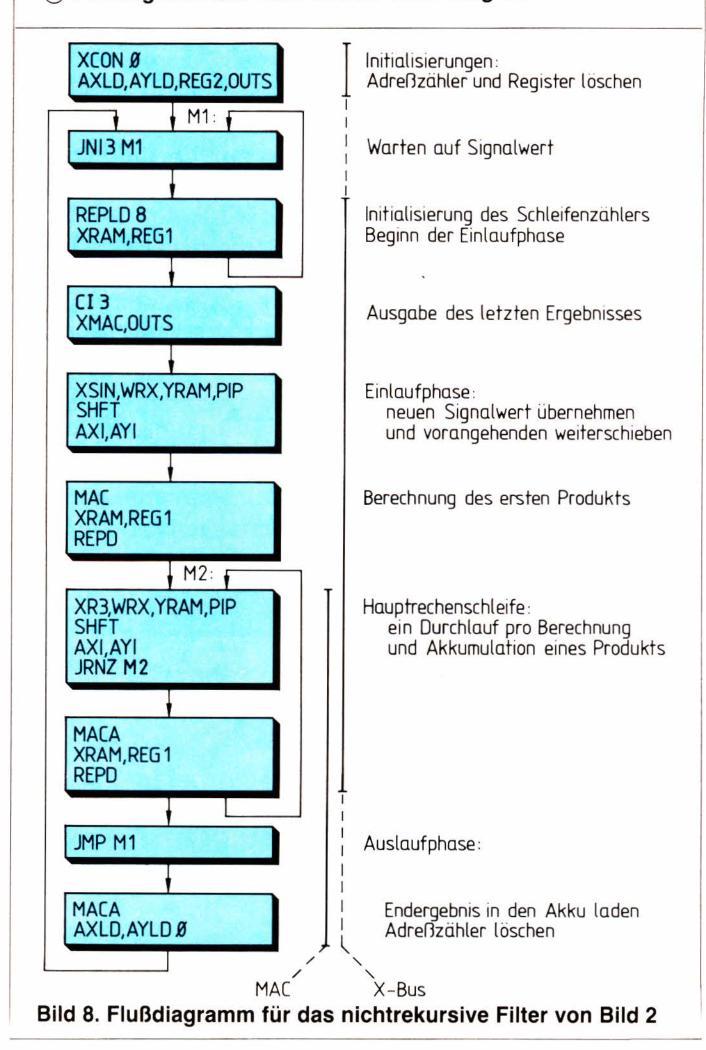


Bild 8. Flußdiagramm für das nichtrekursive Filter von Bild 2

4 Programmbeispiel

Abschließend sei in *Bild 8* als kurzes Programmbeispiel das Flußdiagramm für das nichtrekursive Filter von Bild 2 vorgestellt. Während auf die benutzten Mnemonics nicht näher eingegangen werden soll, sei auf die folgenden Punkte hingewiesen:
– jedes Kästchen entspricht einem Mikrobefehl, dessen Ausführung 80 ns erfordert;
– wegen des Pipelinings der Mikrobefehle werden Sprünge stets erst im übernächsten Zyklus ausgeführt;
– die aus zwei Zyklen bestehende Hauptrechenschleife wird so oft durchlaufen, bis der anfangs mit dem Filtergrad (hier: 8) geladene Repeater durch Dekrementieren den Inhalt Null erreicht hat;
– infolge des Pipelinings der Daten ist eine vom Filtergrad unabhängige Ein- und Auslaufphase nötig, die mit größer werdendem Filtergrad immer weniger ins Gewicht fällt (s. auch Bild 7);
– der X-Bus sowie der Multiplizierer-Akkumulator sind während der Hauptrechenschleife voll ausgelastet. Als Auslastung des MACs bezogen auf die Gesamtrechenzeit des Filters ergeben sich Werte zwischen 44 % (Filtergrad 1) und 99,8 % (Filtergrad 1023).

Für die Unterstützung der Arbeit möchten die Autoren Herrn *Prof. Dr.-Ing. H. W. Schüßler* ihren besonderen Dank aussprechen.

Literatur

[1] *Gold, B., et.al.*: The FDP, a Fast Programmable Signal Processor. IEEE Trans. on Computers, Vol. C-20, No. 1, Jan. 71, S. 33 ff.
[2] *Schloß, H. J., Ballenberger, G.*: A High Speed Small Scale Signal Processor. Signal Processing 3 (1981), S. 29 ff.
[3] *Ebert, P. M., Mazo, J. E., Taylor, M. G.* Overflow Oscillations in Digital Filters. Bell Syst. Techn. J. 48, 1969, S. 2999 ff.
[4] *Rösch, H. P.*: Entwicklung von Signalverarbeitungsprogrammen für den Nano-Signalprozessor. Studienarbeit am Lehrstuhl für Nachrichtentechnik, Univ. Erlangen-Nürnberg, 1982.
[5] *Zeman, J., Lindgren, A. G.*: Fast Digital Filters with Low Roundoff Noise. IEEE Trans. on Circuits and Systems, Vol. 28, Nr. 7, July 1981, 716 ff.

Dr. Sönke Mehrgardt

32-Bit-Prozessor erzeugt analoge Signale

Seit mit monolithischen Signalprozessoren auch zeitkritische Probleme ohne großen Hardware-Aufwand zu lösen sind, bieten sich solche Bausteine auch zur Erzeugung analoger Signale – z. B. in Testsystemen – an. Dieser Beitrag stellt Programme für den 32-Bit-Signalprozessor TMS 320 vor, die Maximalfolgen, Sägezahn- und Rechteckschwingungen, Impulsfolgen sowie Dreieck- und Sinusschwingungen erzeugen. Anhand des Umfanges dieser Routinen können zukünftige Anwender leicht abschätzen, mit welchem Zeitaufwand sie bei der Lösung ihrer eigenen Problemstellung rechnen müssen.

Die dargestellten Programmbeispiele sind hinsichtlich kurzer Ausführungszeiten optimiert. Da dies (erfreulicherweise) gleichzeitig auch zu Programmen führt, die nur wenig Speicher belegen, sind die angegebenen Beispiele oft als Teil größerer Programmeinheiten direkt verwendbar. Die angegebenen Ausführungszeiten gelten für einen Prozessor mit 20 MHz Taktfrequenz, entsprechend einem Befehlszyklus von 200 ns.

1 Maximalfolgen

Bei vielen Meßaufgaben lassen sich mit Vorteil sogenannte Pseudorauschsignale als Testsignal einsetzen. Unter ihnen hat wiederum die Klasse der Maximalfolgen besondere Bedeutung. Die wesentlichen Vorteile dieser Testsignale liegen in der Kombination mehrerer optimaler Eigenschaften:

- Das Signal enthält alle Frequenzen des betrachteten Frequenzbereiches mit möglichst gleicher Energie („weißes Rauschen").
- Das Signal ist exakt reproduzierbar.
- Das Signal nimmt stets einen von zwei extremen Amplitudenwerten an (damit kann ein zu testendes System stets mit voller Amplitude ausgesteuert werden, und das Signal-Rauschverhältnis wird optimal).

Zusätzlich sind in neuerer Zeit Algorithmen bekannt geworden, durch die sich Messungen mit Maximalfolgen-Testsignalen besonders effektiv verarbeiten lassen. Diese Algorithmen basieren auf der *Fast-Hadamard-Transformation* (FHT, [1]). Soll z. B. aus einer Messung die Impulsantwort des Systems berechnet werden, so bieten diese neuen Algorithmen gegenüber der Berechnung mit Hilfe der schnellen Fourier-Transformation (FFT) sowohl geringere Ausführungszeiten, als auch höhere Genauigkeit wegen fehlender Rundungsfehler: Die FHT benötigt nur Additionen und Subtraktionen, *keine* Multiplikationen! Sie ist damit auch besonders leicht auf Mikroprozessoren implementierbar.

Maximalfolgen können durch rückgekoppelte Schieberegister erzeugt werden. Wie *Bild 1* zeigt, werden dazu einige Stufen des Schieberegisters angezapft, addiert (modulo 2, d. h., eventuelle Überträge bleiben

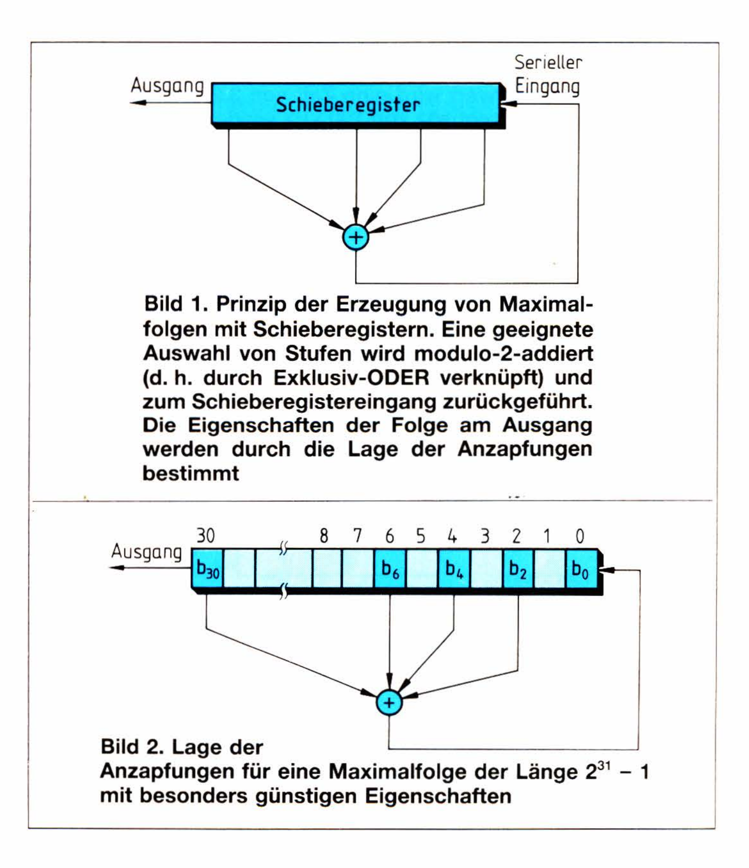


Bild 1. Prinzip der Erzeugung von Maximalfolgen mit Schieberegistern. Eine geeignete Auswahl von Stufen wird modulo-2-addiert (d. h. durch Exklusiv-ODER verknüpft) und zum Schieberegistereingang zurückgeführt. Die Eigenschaften der Folge am Ausgang werden durch die Lage der Anzapfungen bestimmt

Bild 2. Lage der Anzapfungen für eine Maximalfolge der Länge $2^{31} - 1$ mit besonders günstigen Eigenschaften

unberücksichtigt) und zum Schieberegistereingang zurückgeführt.

Soll die Maximalfolge optimale Eigenschaften haben, so müssen möglichst vier oder mehr Anzapfungen verwendet werden. *Bild 2* zeigt die Lage der Anzapfungen zur Erzeugung einer solchen „guten" Maximalfolge mit einem Schieberegister von 31 Stufen. Die Folge hat eine Periodenlänge von $2^{31} - 1$, das sind 2 147 483 647 Abtastwerte. Bei einer Abtastfrequenz von 100 kHz wiederholt sich das Signal damit erst nach etwa sechs Stunden.

Speicherbelegung:

NAME:	KONST	SCRA	RH RL
INHALT:	KONSTANTE 8000H	ZWISCHEN-SPEICHER	SCHIEBEREGISTER

Programm:

```
LAC    1, RH     BIT 31
AND    KONST
ADD    9, RL     ⊕ BIT 6
AND    KONST
ADD    11, RL    ⊕ BIT 4
AND    KONST
ADD    13, RL    ⊕ BIT 2
AND    KONST
SACH   1, SCRA
LAC    0, SCRA   (ACC) = BIT 31 ⊕ BIT 6 ⊕ BIT 4 ⊕ BIT 2
ADDS   RL
ADDS   RL             + 2* (RL)
ADDH   RH
ADDH   RH                  + 2* (RH)
SACL   0, RL
SACH   0, RH
```

Bild 3. Programm zur Berechnung der Maximalfolge in Bild 2. Je Abtastwert wird eine Zeit von 3,2 µs benötigt. Bei einer Abtastfrequenz von 100 kHz wiederholt sich die Folge erst nach sechs Stunden

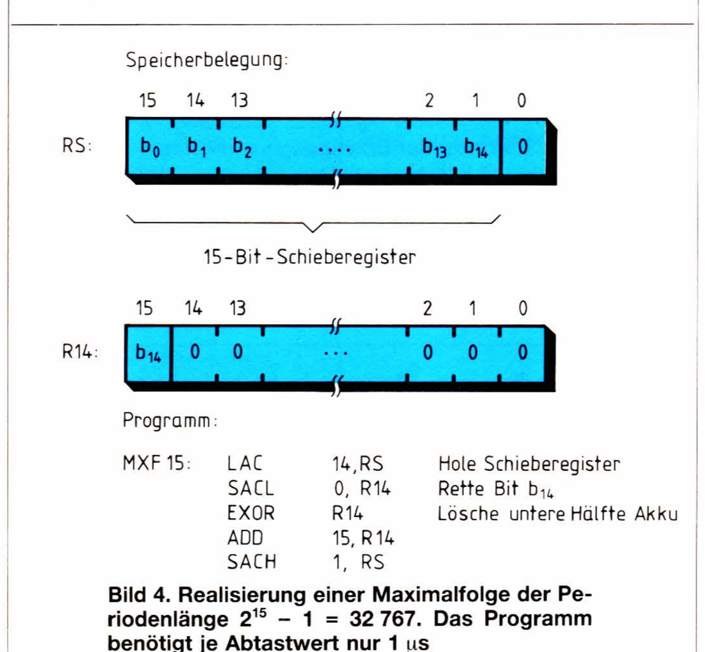


Bild 4. Realisierung einer Maximalfolge der Periodenlänge $2^{15} - 1 = 32\,767$. Das Programm benötigt je Abtastwert nur 1 µs

Ein Programm zur Erzeugung dieser Maximalfolge mit dem TMS 320 zeigt *Bild 3*. Die vom Programm benutzten internen Datenspeicher sind ebenfalls angegeben. Zur Berechnung eines neuen Folgenwertes benötigt das Programm 16 Zyklen, entsprechend 3,2 µs. Damit lassen sich mit dem Prozessor auch noch bei hohen Abtastfrequenzen Messungen durchführen, bei denen der Prozessor in Echtzeit das Testsignal berechnet und gleichzeitig die Meßdaten erfaßt.

Werden die Eigenschaften und die große Periodenlänge dieser Folge nicht benötigt, so lassen sich einfachere Folgen verwenden. *Bild 4* enthält die nötigen Angaben zur Realisierung einer Maximalfolge der Länge $2^{15} - 1$ mit Anzapfungen der ersten und der letzten Schieberegisterstufen. Dieses Programm benötigt zur Berechnung eines neuen Folgenwertes nur noch 1 µs. Ermöglicht wird die geringe Rechenzeit durch die Nutzung der im Befehlssatz vorhandenen Möglichkeiten zum Daten-Schieben bei den Speicherzugriffen.

2 Sägezahn, Rechteck, Dreieck

Sägezahnschwingungen haben nicht nur Bedeutung als Testsignale. Sie werden oft auch bei elektronischer Musik eingesetzt, da ein Sägezahn alle Harmonischen enthält, so daß sich durch einfache Filterung jede gewünschte Klangfarbe erzeugen läßt.

Den einfachsten digitalen Sägezahngenerator stellt ein Binärzähler dar, der im Takt der Abtastfrequenz erhöht wird. Wählt man z. B. einen 8stufigen Binärzähler aus, so liefern die Ausgänge des Zählers gerade ein Sägezahnsignal von $1/256$ der Abtastfrequenz. Sollen jedoch bei fest vorgegebener Abtastfrequenz beliebige Frequenzen des Sägezahnsignals einstellbar sein, so ist eine leichte Variation dieses Prinzips nötig: Der Zähler wird nun nicht einfach inkrementiert, sondern im Rhythmus der Abtastfrequenz um einen bestimmten Zahlenwert erhöht. Man benutzt dann beispielsweise ein 16-Bit-Datenwort und erhöht es in jedem Abtastintervall um eine Zahl N (dies ist mit jedem Mikroprozessor sehr leicht möglich). Der Algorithmus lautet also:

$$(\text{Daten}) + N \rightarrow (\text{Daten})$$

Setzen wir der Einfachheit halber eine Abtastfrequenz von $2^{16} = 65\,536$ Hz voraus, so wird bei einem Wert N = 1 die volle Sägezahnperiode gerade in einer Sekunde durchlaufen. Entsprechend dauert es bei N = 2 dann 0,5 s usw. Die Zahl N entspricht also gerade der Frequenz der Sägezahnschwingung (in Hz). Damit läßt sich bei fester Abtastfrequenz die Sägezahnfrequenz in 1-Hz-Stufen über einen weiten Bereich ändern.

Das Programm auf dem TMS 320 ist sehr einfach und benötigt je Abtastwert nur 0,6 µs (*Bild 5*). Weitere Verfeinerungen sind denkbar, wenn mit noch höherer Genauigkeit (z. B. 32 Bit) gerechnet wird. Die Frequenzauflösung steigt dann erheblich an.

Ein generelles Problem bei der Erzeugung solcher nicht-bandbeschränkter Signale ist es, daß durch die

Verletzung des Abtast-Theorems Fehler auftreten. Im vorliegenden Fall kann man sich dies wie folgt vergegenwärtigen: Die berechneten Zahlenwerte eines Sägezahngenerators nach Bild 5 sind gerade die Werte, die man bei Abtastung einer idealen Sägezahnschwingung der gewünschten Frequenz mit der gegebenen Abtastrate erhielte. Da die ideale Sägezahnschwingung nicht bei der halben Abtastfrequenz tiefpaßgefiltert wurde, sondern auch Spektrallinien bei höheren Frequenzen enthält, gibt es störende Überlagerungen (*Aliasing*). Wegen der proportional zur Frequenz absinkenden Amplitude der Sägezahn-Harmonischen stört dieser Effekt natürlich um so weniger, je tiefer die Sägezahnfrequenz im Vergleich zur Abtastfrequenz ist.

```
SEGN:   ZALS    SAEGE
        ADDS    FREQ
        SACL    0,SAEGE
```

Bild 5. Programm für eine Sägezahnschwingung: Der Speicherplatz SAEGE enthält das Signal und FREQ enthält den Frequenzwert N. Die Berechnung eines Abtastwertes erfordert nur 0,6 µs

```
SPRGEN: ZALS    SAEGE
        SACH    1,RECT      RECHTECKSIGNAL
        ADDS    FREQ
        SACL    0,SAEGE     SAEGEZAHNSIGNAL
        SACH    0,PULS      UEBERLAUF GIBT PULS
```

Bild 6. Durch Hinzufügen zweier Befehle wird aus dem Sägezahngenerator (Bild 5) zusätzlich ein Rechteck- und ein Pulsgenerator. Ein Abtastintervall dauert nur 1 µs

Aus dem Sägezahngenerator lassen sich leicht eine Rechteckschwingung und eine Impulsfolge ableiten. So ergibt beispielsweise bei dem in Bild 5 dargestellten Sägezahngenerator das höchstwertige Bit der 16 Datenbits gerade die Rechteckschwingung, während die Verwendung des Überlaufes bei der Addition zu einer Impulsfolge führt. Ein Programm, das die drei Schwingungsformen erzeugt, ist in *Bild 6* dargestellt. Durch geeignete Ausnutzung des TMS-320-Befehlssatzes ist es möglich, die drei je Abtastintervall benötigten Werte in nur 1 µs (!) zu berechnen. Diese Signalformen können damit auch in zeitkritischen Anwendungen meist noch in Echtzeit erzeugt werden.

Eine weitere Schwingungsform, die leicht aus dem Sägezahn herzuleiten ist, ist die Dreieckschwingung. Dieses Signal ist in manchen Anwendungen vorteilhaft, in denen ein geringer Oberwellengehalt erwünscht ist: Während die Sägezahnschwingung alle Oberwellen enthält (mit Amplituden proportional zu 1/f), besteht die Dreieckschwingung nur aus den ungeraden Oberwellen, deren Amplituden außerdem wesentlich schneller, proportional zu $1/f^2$, fallen.

Aus der Sägezahnschwingung läßt sich die geeignet skalierte Dreieckschwingung mit nur drei Befehlen berechnen (*Bild 7*). Das Programm benötigt gegenüber dem Sägezahngenerator zusätzlich 0,6 µs an Ausführungszeit.

In *Bild 8* sind vier Signale dargestellt, die mit den in diesem Abschnitt dargestellten Programmteilen generiert werden können. Als Frequenzwert wurde N = 6000 gewählt, entsprechend einer Periodenlänge der Schwingungen von $2^{16}/6000 = 10{,}9$ (Abtastintervalle). Die erwähnten Probleme bei digitaler Erzeugung nichtbandbeschränkter Signale führen bei den vier Zeitfunktionen dazu, daß die Periodenlänge um ein Abtastintervall schwankt: Die mittlere Periode in Bild 8 hat eine Länge von nur zehn Abtastintervallen, während die übrigen Perioden sich über elf Intervalle erstrecken. In der Praxis stört dieses Schwanken (*Jitter*) der Periodenlänge dann relativ wenig, wenn die Frequenz der Schwingungen ausreichend tief gegen die Abtastfrequenz ist.

Besonders eindrucksvoll lassen sich die durch Überlagerung hoher Harmonischer hervorgerufenen Fehler mit den Spektren der Signale demonstrieren. Zur Illustration zeigen die *Bilder 9* und *10* Spektren der Sägezahn- bzw. der Dreieckschwingung (obere bzw. untere Kurve

Speicherbelegung:

NAME:	SAEGE	FREQ	OFFSET	DREI
INHALT:	Sägezahn	Frequenz	4000 H	Dreieck

Programm:

```
DRGEN:  ZALH    SAEGE
        ADDH    FREQ
        SACH    0, SAEGE    Sägezahn berechnet
        ABS                 Absolutbetrag davon
        SUBH    OFFSET      minus 4000H
        SACH    1, DREI     gibt Dreieckschwingung
```

Bild 7. Das Programm erzeugt eine Dreieckschwingung aus einer Sägezahnschwingung. Die in 1,2 µs (je Abtastwert) berechnete Dreieckschwingung ist besonders oberwellenarm

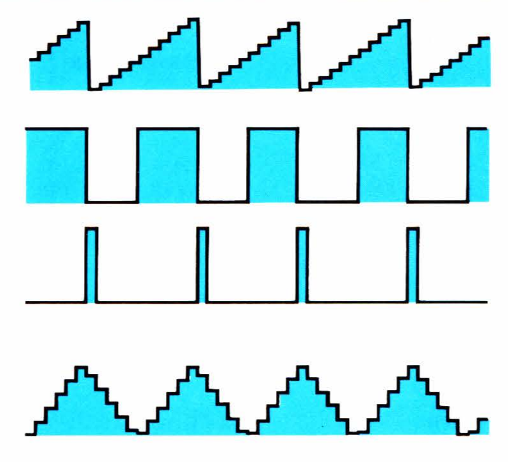


Bild 8. Kurvenformen für einen Frequenzwert N = 6000. Die Frequenz der Schwingungen beträgt also $6000/2^{16}$ · Abtastfrequenz

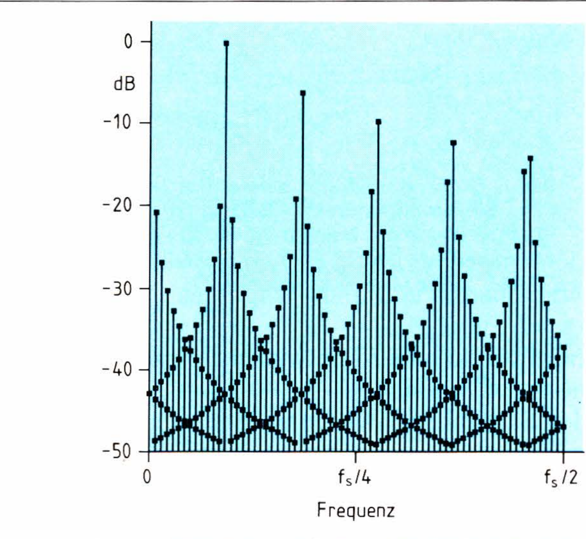


Bild 9. Spektrum der Sägezahnschwingung aus Bild 8 (obere Kurve). Die große Anzahl von Spektrallinien entsteht durch Überlagerung (Aliasing) höherfrequenter Harmonischer

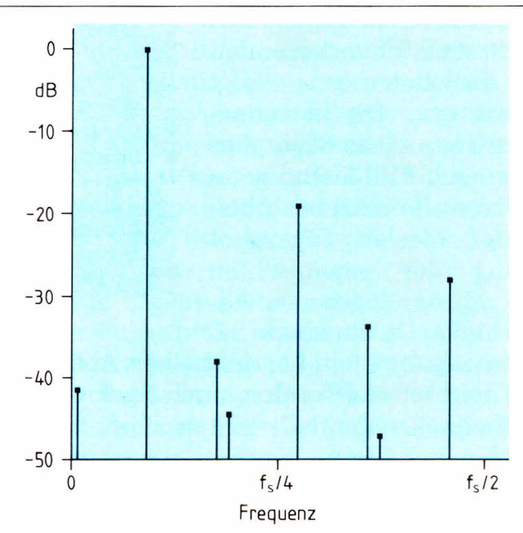


Bild 10. Spektrum der Dreieckschwingung aus Bild 8 (untere Kurve). Gegenüber einer Sägezahnschwingung fehlen die geraden Harmonischen völlig, und die Amplituden der Spektrallinien fallen wesentlich schneller ab

in Bild 8). Beim Spektrum der Sägezahnschwingung gehören die fünf höchsten Spektrallinien zu den Harmonischen, die unterhalb der halben Abtastfrequenz liegen. Alle weiteren Spektrallinien kommen durch die Überlagerung der höherfrequenten Harmonischen zustande. Ihre Anzahl ist, wegen der nur langsam fallenden Amplituden der Teiltöne, sehr hoch.

Das wesentlich schneller abfallende Amplitudenspektrum der Dreieckschwingung zeigt Bild 10. Da außerdem die geraden Harmonischen entfallen, liegen im dargestellten Amplitudenbereich (bis −50 dB) nur noch acht Spektrallinien. Wegen der relativ kleinen Störungen durch überlagerte höherfrequente Harmonische und wegen des Amplitudenabstandes zwischen der Grundwelle und der nächsten Harmonischen von 19 dB läßt sich aus diesem Signal leicht eine Sinusschwingung berechnen.

3 Sinusgeneratoren

Digitale Sinusgeneratoren können besonders gute Eigenschaften erreichen [2]. Welche Möglichkeiten bietet dabei der Prozessor TMS 320?

Der Befehlssatz des TMS 320 unterstützt sehr effektiv die Realisierung digitaler Filter. Es ist daher naheliegend, durch Bandpaßfilterung z. B. einer Sägezahn- oder Dreieckschwingung die Sinusschwingung zu generieren.

Ein einfacher digitaler Bandpaß der Mittenfrequenz $G = 2\pi f_{Mitte}/f_{Abtast}$ wird beschrieben durch die Differenzengleichung

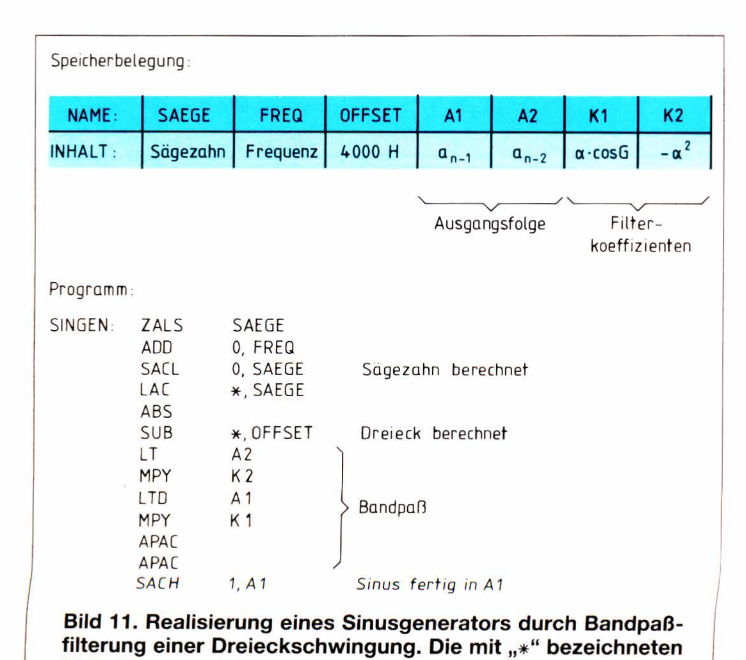
$$a_n = e_n + (2\alpha \cdot \cos G) \cdot a_{n-1} - \alpha^2 \cdot a_{n-2} \; (\alpha = 0{,}8...0{,}99)$$

Dabei ist e_n die Eingangsfolge des Filters und a_n die Ausgangsfolge. Der Wert α bestimmt die Güte des Bandpasses, er sollte im Bereich 0,8...0,99 liegen.

Verwendet man den im vorigen Abschnitt beschriebenen Dreieckgenerator zur Speisung dieses Filters und drückt die Mittenfrequenz G durch die Sägezahnfrequenz N (siehe Bild 5) aus, bei einem Wert $\alpha = 0{,}99$, so ergibt sich die „Dimensionierung"

$$a_n = e_n + 1{,}98 \cdot \cos(\pi N/32\,768) \cdot a_{n-1} - 0{,}98 \cdot a_{n-2}$$

Das gesamte Programm eines Sinusgenerators nach diesem Prinzip zeigt *Bild 11*. Man erkennt in den ersten drei Befehlen den Sägezahngenerator wieder, während


Speicherbelegung:

NAME:	SAEGE	FREQ	OFFSET	A1	A2	K1	K2
INHALT:	Sägezahn	Frequenz	4000 H	a_{n-1}	a_{n-2}	$\alpha \cdot \cos G$	$-\alpha^2$

Programm:

```
SINGEN: ZALS   SAEGE
        ADD    0, FREQ
        SACL   0, SAEGE      Sägezahn berechnet
        LAC    *, SAEGE
        ABS
        SUB    *, OFFSET     Dreieck berechnet
        LT     A2          )
        MPY    K2          |
        LTD    A1          |
        MPY    K1          } Bandpaß
        APAC               |
        APAC               )
        SACH   1, A1         Sinus fertig in A1
```

Bild 11. Realisierung eines Sinusgenerators durch Bandpaßfilterung einer Dreieckschwingung. Die mit „*" bezeichneten Shift-Werte sind je nach der Sinusfrequenz so zu wählen, daß sich eine optimale Amplitude ergibt

die übrigen zehn Befehle die Umwandlung in die Dreieckschwingung und den digitalen Bandpaß realisieren. Die Berechnung eines Abtastwertes ist damit in nur 2,6 µs vollzogen.

Je nach der eingestellten Frequenz der Sinusschwingung liegen im dargestellten Beispiel die Amplituden der Oberwellen etwa 40...50 dB unter der Nutzschwingung. Für viele Anwendungen ist dieser Wert völlig ausreichend.

Für sehr hohe Ansprüche muß aber die Sinusfunktion wirklich genau berechnet werden. Den schnellsten Weg bieten natürlich abgespeicherte Sinustafeln. Jedoch steht oft der dazu benötigte große Speicherbereich nicht zur Verfügung. Man kann sich dann mit kleineren Tabellen helfen, in denen geeignet interpoliert werden muß, um die gewünschte Genauigkeit zu erreichen. So liefert beispielsweise die lineare Interpolation einer Tabelle der Länge 256 (für eine abgespeicherte Sinusperiode) bereits die Sinusfunktion auf 15 Bit genau.

Eine Untersuchung dieser Möglichkeiten hat jedoch ergeben, daß sich eine schnellere und weniger speicherintensive Realisierung der Sinusfunktion durch Polynomapproximationen ergibt. Die Sinusfunktion wird bekanntlich durch die Reihe

$$\sin(x) = x - x^3/3! + x^5/5! - x^7/7! + \ldots$$

dargestellt. Trifft man einen Kompromiß zwischen Genauigkeit und Geschwindigkeit, so reicht eine Approximation 5ten Grades aus: *Bild 12* zeigt den Fehler der Approximation für Werte des Argumentes zwischen 0 und $\pi/2$. Während der Fehler der bei x^5 einfach abgebrochenen Reihe schnell wächst, läßt sich durch eine Optimierung der Koeffizienten

$$\sin(x) \approx 0{,}9997x - 0{,}165629x^3 + 0{,}0074886x^5$$

im dargestellten Argumentbereich eine 14-Bit-Genauigkeit erreichen.

Zur Umsetzung dieser Formel in ein TMS-320-Programm wird die übliche Argument-Skalierung angenommen, bei der der Wertebereich eines 16-Bit-Zweierkomplement-Bruches (−1...0,99997) gerade auf den Bereich $-\pi$ bis $+\pi$ abgebildet wird. Stellt man Argumente in dieser Normierung dar, so entspricht dem Überlauf des Wertebereiches gerade die gewünschte Operation „modulo 2π". Die Formel zur Berechnung der Sinusfunktion lautet jetzt

$$\sin(\pi x) = 3{,}1407x - 5{,}1355x^3 + 2{,}2917x^5$$

wobei x nun ein 16-Bit-Zweierkomplement-Bruch ist. Im Hinblick auf den Befehlssatz des TMS 320 muß diese Formel nochmals umskaliert werden. Gleichzeitig verwendet man das Horner-Schema, um die Anzahl der Rechenschritte zu verkleinern. Die Formel lautet nun

$$\sin(\pi x) = 8x \left\{ \frac{12\,864}{32\,786} + x^2 \left(\frac{-21\,038}{32\,768} + \frac{9\,387}{32\,768}\, x^2 \right) \right\}$$

Durch Ausklammern des Faktors 8 wird hier erreicht, daß nur noch Koeffizienten auftreten, die als 16-Bit-Bruch darstellbar sind.

Die angegebene Polynomapproximation wird nur für Argumente im Bereich $-\pi/2$ bis $\pi/2$ direkt verwendet. Die verbleibenden beiden Teilbereiche ($-\pi$, $-\pi/2$ und $\pi/2$,π) werden durch eine geeignete Argumentumrechnung auf diesen Bereich zurückgeführt. Im Sinusprogramm (*Bild 13*) werden dazu die ersten sechs Befehle

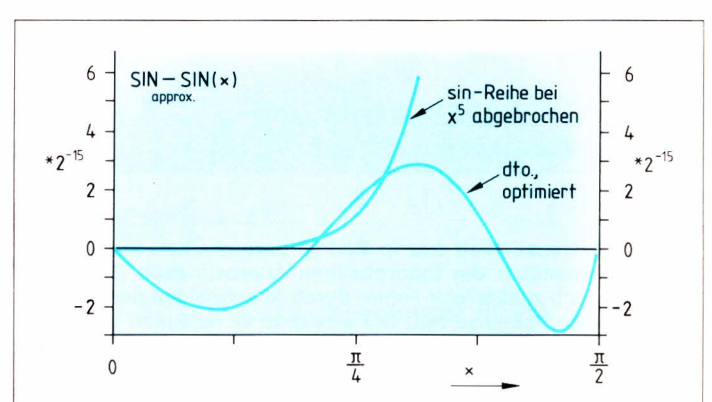


Bild 12. Approximation der Sinus-Funktion durch Polynome. Bei einfachem Abbrechen der Sinus-Reihe bei x^5 ergibt sich für große Argumente ein großer Fehler. Er läßt sich durch Optimierung der Approximation wesentlich verringern

Speicherbelegung:

SPEICHERNAME	ZWECK
RX	ARGUMENT DER SINUSFUNKTION MIT DER NORMIERUNG: 4000H ≙ π/2
RSIN	ERGEBNIS
R9387	KONSTANTE 9387 (=KOEFFIZIENT 0,28646)
R21038	KONSTANTE 21038 (=KOEFFIZIENT 0,64194)
R12864	KONSTANTE 12864 (=KOEFFIZIENT 0,39258)
RMASK	KONSTANTE 8000H

Programm:

```
SIN5:   LAC     1, RX
        EXOR    RX
        AND     RMASK
        BZ      SIN 50
        SUBS    RX
        SACL    0, RX
SIN50:  LT      RX
        MPY     RX
        PAC
        SACH    1, RSIN
        LT      RSIN
        MPY     R9387
        PAC
        SUB     15, R21038
        SACH    1, RSIN
        MPY     RSIN
        LAC     15, R12864
        LTA     RX
        SACH    1, RSIN
        MPY     RSIN
        PAC
        SACH    4, RSIN
        (RET)
```

Bild 13. Programm zur Berechnung der Sinusfunktion. Je nach Wert des Argumentes werden 4,2 µs oder 4,6 µs benötigt

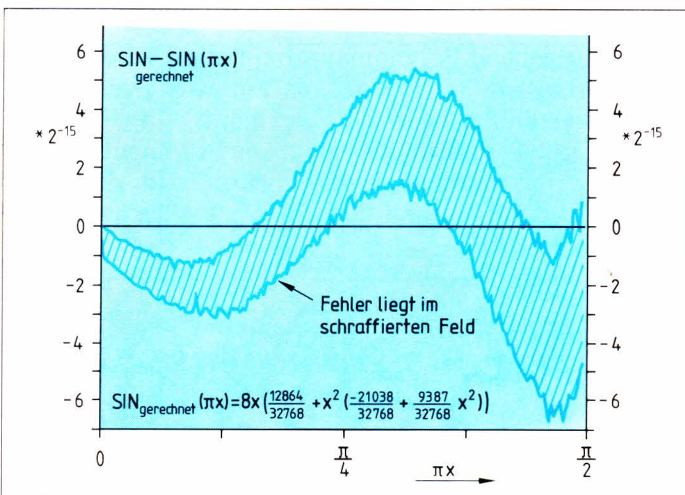


Bild 14. Genauigkeit des in Bild 13 dargestellten Programmes. Gegenüber der theoretischen Approximation (Bild 12) treten noch zusätzliche Fehler durch Abschneiden der Multiplikationsergebnisse auf. Die erreichte Genauigkeit beträgt rund 13,5 Bit

benötigt. Die eigentliche Polynomapproximation umfaßt dann weitere 16 Befehle. Zur Berechnung eines Sinuswertes werden also insgesamt 4,2 µs bzw. 4,6 µs benötigt, je nach Wert des Argumentes.

Abschließend soll erneut die Genauigkeit des Sinus-Programmes betrachtet werden. Bei der Realisierung tritt durch das Abschneiden niederwertiger Bits der 32-Bit-Produkte ein zusätzlicher Fehler auf, der die ursprüngliche Approximation verschlechtert und außerdem zu einem Überlauf bei den Rechnungen führen kann. Die vom dargestellten Programm schließlich erreichte Genauigkeit zeigt *Bild 14*. Aufgrund der angesprochenen Probleme ist der Fehler nun, verglichen mit Bild 12, etwas gewachsen, und die Sinusfunktion wird mit einer Genauigkeit von etwa 13,5 Bit errechnet.

Zur Realisierung eines digitalen Sinusgenerators läßt sich die entsprechende Zahlenfolge

$$s_n = \sin(2\pi n f_{Sinus}/f_{Abtast})$$

nun leicht berechnen: Das Argument der Sinusfunktion liefert gerade der schon dargestellte Sägezahngenerator. Bei einer 32-Bit-Rechengenauigkeit werden dazu 1,2 µs benötigt. Weitere 4,6 µs erfordert das Sinusprogramm. Addiert man noch die Zeit für die Ausgabe des berechneten Wertes und die Programmschleife, so ergibt sich je Abtastwert eine Rechenzeit von nur 6,6 µs.

Der Prozessor kann damit entweder mit der maximalen Abtastfrequenz von 151 kHz Sinusschwingungen hoher Qualität (über 80 dB Nebenwellenabstand, Frequenzbereich 0...70 kHz, Frequenzauflösung 35 µHz!) erzeugen, oder aber man nutzt bei geringeren Abtastfrequenzen die verbleibende Rechenkapazität für Auswertungen oder Nebenrechnungen aus.

Literatur

[1] *Alrutz, H.:* Ein neuer Algorithmus zur Auswertung von Messungen mit Pseudorauschsignalen. Fortschritte der Akustik, DAGA '81 Berlin, VDE-Verlag Berlin, S. 525...528.

[2] *Mehrgardt, S., Alrutz, H.:* Digitaler Sinusgenerator hoher Präzision. ELEKTRONIK 1983, H. 5, S. 53...57.

[3] 32-Bit-Mikrocomputer für Signalverarbeitung und Prozeßsteuerung. ELEKTRONIK 1983, H. 22, S. 139...141.

[4] TMS 320 Preliminary Functional Specification. Texas Instruments.

[5] *Mehrgardt, S.:* Universelles Prozessorsystem zur digitalen Signalverarbeitung. ELEKTRONIK 1984, H. 3, S. 49...53.

Dipl.-Ing. (FH) Thomas Dischinger
Prof. Dr.-Ing. Horst Nielinger

Linearphasige FIR-Filter mit Signalprozessor realisiert

FIR-Filter lassen sich mit Signalprozessoren relativ leicht realisieren. In [1] ist für den Signalprozessor µPD 7720 (NEC) das Programm für ein 64gradiges FIR-Filter angegeben. In der Praxis reicht dieser Grad häufig nicht aus. Daher beschreibt dieser Beitrag ein Programm für den Signalprozessor µPD7220, mit dessen Hilfe ein linearphasiges Filter bis zum Grad N = 128 realisiert werden kann.

Bild 1 zeigt das Blockdiagramm des Signalprozessors µPD 7720. Während das in [1] angegebene Programm für das 64gradige FIR-Filter die Abtastwerte und die Koeffizienten im RAM speichert, wird hier das gesamte RAM für die Abtastwerte benutzt, wobei die Koeffizienten im Daten-ROM abgelegt werden, das allerdings nur 13 Bit breit ist.

Bild 2 zeigt die Struktur eines linearphasigen FIR-Filters 128. Grades. Die hexadezimalen RAM-Adressen für die einzelnen Abtastwerte sind angegeben.

Die *Tabelle* zeigt die zugehörige Speicherbelegung. Das gewählte Schema hängt mit den in [1] näher beschriebenen Manipulationsmöglichkeiten der drei höheren Bits des „Data Pointers" (DP_H) zusammen. So erlaubt es die Anweisung „M 4" z. B., die RAM-Adresse 40H in 00H zu ändern und damit die in der FIR-Filter-Struktur (Bild 2) gegenüberliegenden Abtastwerte X_0 und X_{127} sehr einfach zu verknüpfen.

Das Filterprogramm

Es soll hier nicht auf jeden Befehl eingegangen werden. Mit Hilfe der Kommentare und der in [1] gegebenen Informationen über den Befehlssatz des Signalprozes-

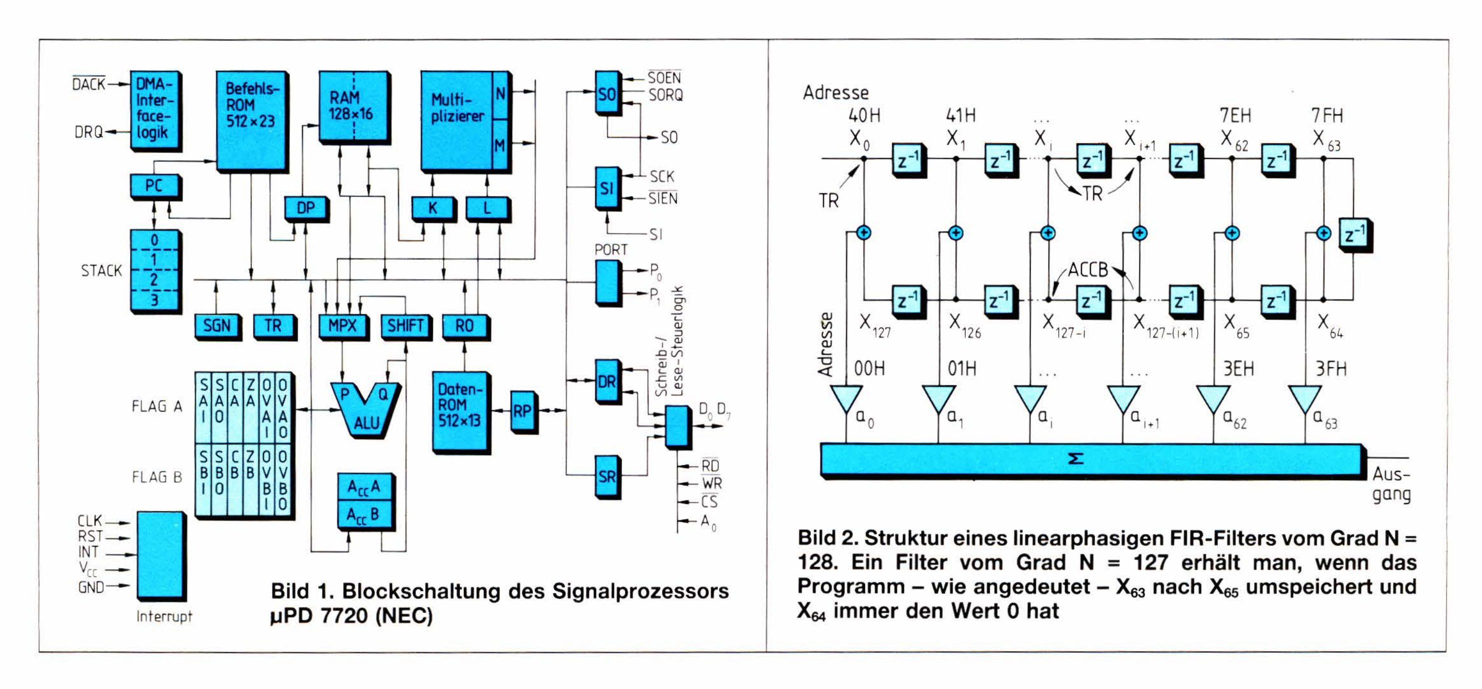


Bild 1. Blockschaltung des Signalprozessors µPD 7720 (NEC)

Bild 2. Struktur eines linearphasigen FIR-Filters vom Grad N = 128. Ein Filter vom Grad N = 127 erhält man, wenn das Programm – wie angedeutet – X_{63} nach X_{65} umspeichert und X_{64} immer den Wert 0 hat

```
                  **  UPD7720  ASSEMBLY LIST  ** (24.3.83   ) MODULE              PAGE 0001

                 (FILTER BIS ZU 128.GRADES        )

STNO LOC.   SOURCE STATEMENT

0001 0000   /*              ******************************************************
0001 0000                   *                                                    *
0001 0000                   *                                                    *
0001 0000                   *                FILTERPROGRAMM                      *
0001 0000                   *                                                    *
0001 0000                   *                                                    *
0001 0000                   ******************************************************
0001 0000
0001 0000
0001 0000             DIESES FILTERPROGRAMM ERMOEGLICHT ES FILTER BIS ZU 128.GRADES
0001 0000             MIT DEM SIGNALPROZESSOR UPD 7720 ZU REALISIEREN. UM DEN FILTER-
0001 0000             GRAD ZU VERAENDERN GIBT ES ZWEI MOEGLICHKEITEN:
0001 0000
0001 0000             1) BEIM FESTLEGEN DER FILTERKOEFFIZIENTEN DIE NICHT BENOETIGTEN
0001 0000                KOEFFIZIENTEN NULL SETZEN. DADURCH WIRD DAS FILTER ZWAR
0001 0000                WIE EIN FILTER 128.GRADES BEARBEITET, JEDOCH WIRD DURCH DAS
0001 0000                NULLSETZEN DER FILTERKOEFFIZIENTEN DER GRAD AUTOMATISCH
0001 0000                VERRINGERT.
0001 0000
0001 0000             2) DURCH VERAENDERUNG DER KONSTANTEN FILGRD IM KOPF DIESES
0001 0000                PROGRAMMS. BEI HERABSETZUNG DES FILTERGRADES UNTER 96
0001 0000                IST ES DANN JEDOCH NOTWENDIG, BESTIMMTE PROGRAMMTEILE ZU
0001 0000                LOESCHEN. DIESE PROGRAMMTEILE WERDEN IM PROGRAMM NAEHER
0001 0000                BESCHRIEBEN. BEI FILTERN UNTER 64. GRADES EMPFIEHLT ES SICH
0001 0000                WEGEN DER HOEHEREN VERARBEITUNGSGESCHWINDIGKEIT DAS PROGRAMM
0001 0000                DER FA. NEC ZU BENUTZEN. FALLS DIESES NICHT VORLIEGEN SOLLTE
0001 0000                KANN EIN SOLCHES FILTER NATUERLICH AUCH MIT DIESEM PROGRAMM
0001 0000                REALISIERT WERDEN.
0001 0000
0001 0000
0001 0000             NORMALERWEISE IST DIE METHODE 2 VORZUZIEHEN, WEIL SICH DAMIT DIE
0001 0000             VERARBEITUNGSGESCHWINDIGKEIT DES FILTERS Z.T. ERHEBLICH VER-
0001 0000             GROESSERT, WAS ZUR FOLGE HAT, DASS HOEHERE ABTASTRATEN BENUTZT
0001 0000             WERDEN KOENNEN.
0001 0000
0001 0000
0001 0000
0001 0000   KONSTANTENDEFINITIONEN :
0001 0000   */
0001 0000   MPROG;
0002 0000   FILGRD   EQUAL    128D        ;/* FILTERGRAD DES FILTERS */
0003 0000   ROMEND   EQUAL 100H+FILGRD/2-1 ;/* OBERSTE BENOETIGTE ADRESSE IM ROMBEREICH */
0004 0000   RAMBEG   EQUAL 128D-FILGRD/2  ;/* STARTADRESSE IM RAMBEREICH */
0005 0000   LINMIU   EQUAL    0E6H        ;/* STANDARDADRESSE DER LINMIU ROUTINE */
0006 0000   /*
0006 0000
0006 0000
0006 0000         PROGRAMMSTART
0006 0000
0006 0000
0006 0000   */

                  **  UPD7720  ASSEMBLY LIST  ** (24.3.83   ) MODULE              PAGE 0002

                 (FILTER BIS ZU 128.GRADES        )

STNO LOC.   SOURCE STATEMENT

0006 0000   ORG 0C0H  ;
0007 00C0   FILTER:    LDI @RP,ROMEND     ;/* SETZEN DES ZEIGERS AUF ERSTE KONSTANTE */
0008 00C1              LDI @DP,RAMBEG     ;/* SETZEN DES ZEIGERS AUF RAMBEG */
0009 00C2              CALL LOOP          ;/* BEARBEITUNG DER ERSTEN 16 FILTERWERTE */
0010 00C3   /*
0010 00C3
0010 00C3    FUER EIN FILTER <= 96.GRADES DIE FOLGENDEN 3 STATEMENTS WEGLASSEN.
0010 00C3
0010 00C3   */
0010 00C3              OP MOV @B,MEM      /* ADRESSE 10H      X(111)---> ACCB  */
0010 00C3                 DPDEC           /* ADRESSE 1FH  */
0010 00C3                 M1              ;/* ADRESSE 0FH  */
0011 00C4              OP MOV @MEM,B      /* X(111) UEBERSCHREIBT X(112) */
0011 00C4                 DPINC           /* ADRESSE 00H  */
0011 00C4                 M5              ;/* ADRESSE 50H  */
0012 00C5              CALL LOOP          ;/* BEARBEITEN DER NAECHSTEN 16 FILTERKOEFF. */
```

```
0013 00C6  /*
0013 00C6
0013 00C6   FUER EIN FILTER <= 64.GRADES DIE FOLGENDEN 4 STATEMENTS WEGLASSEN.
0013 00C6
0013 00C6  */
0013 00C6             OP M2              ;/* ADRESSE 00H --- ADRESSE 20H */
0014 00C7             OP MOV @B,MEM       /* X(95) --) ACCB */
0014 00C7                DPDEC            /* ADRESSE 2FH */
0014 00C7                M3              ;/* ADRESSE 1FH */
0015 00C8             OP MOV @MEM,B       /* X(95) UEBERSCHREIBT X(96) */
0015 00C8                DPINC            /* ADRESSE 10H */
0015 00C8                M7              ;/* ADRESSE 60H */
0016 00C9             CALL LOOP          ;/* BEARBEITEN DER NAECHSTEN 16 FILTERKOEFF.*/
0017 00CA  /*
0017 00CA
0017 00CA  FUER EIN FILTER <= 32.GRADES DIE FOLGENDEN 3 STATEMENTS WEGLASSEN
0017 00CA
0017 00CA  */
0017 00CA             OP MOV @B,MEM       /* ADRESSE 30H       X(79)--) ACCB */
0017 00CA                DPDEC            /* ADRESSE 3FH */
0017 00CA                M1              ;/* ADRESSE 2FH */
0018 00CB             OP MOV @MEM,B       /* X(79) UEBERSCHREIBT X(80) */
0018 00CB                DPINC            /* ADRESSE 20H */
0018 00CB                M5              ;/* ADRESSE 70H */
0019 00CC                CALL LOOP       ;/* BEARBEITEN DER NAECHSTEN 16 FILTER KOEFF */
0020 00CD  /*
0020 00CD
0020 00CD
0020 00CD  */
0020 00CD                OP MOV @RP,SIM   /* EINLESEN DES NEUEN ABTASTWERTES */
0020 00CD                   DPDEC         /* ADRESSE 2FH */
0020 00CD                   M1           ;/* ADRESSE 3FH */
0021 00CE                CALL LINMIU     ;/* FILTERERGEBNIS LINMIU WANDELN */
0022 00CF                OP MOV @SOM,A    /* UND AUSGEBEN */
0022 00CF                   XOR ACCA,IDB;/* ACCA LOESCHEN */
0023 00D0  /*               DPDEC            FUER N UNGERADE  ADRESSE 3EH
0023 00D0
0023 00D0
0023 00D0  */

           **  UPD7720  ASSEMBLY LIST  ** (24.3.83  ) MODULE              PAGE 0003
            (FILTER BIS ZU 128.GRADES         )

STNO LOC.  SOURCE STATEMENT

0023 00D0                OP MOV @MEM,TR ;/* X(63) UEBERSCHREIBT X(64) */
0024 00D1  /*               DPINC            FUER N UNGERADE: ADRESSE 3FH   */
0024 00D1                LDI @MEM,0000H ;/* FUER N UNGERADE: 000H IN 3FH */
0025 00D2                OP MOV @TR,RO  ;/* HOLEN DES DECODIERTEN EINLESEWERTES */
0026 00D3                LDI @SR,0380H  ;/* ENABLE INTERUPT */
0027 00D4                JMP $          ;/* WARTEN AUF DEN INTERUPT */
0028 00D5  /*
0028 00D5  *************************************************************************
0028 00D5
0028 00D5                UNTERPROGRAMM LOOP
0028 00D5                IN DIESEM UNTERPROGRAMM WIRD JEDER EINZELNE EINLESWERT
0028 00D5                MIT DER ENTSPRECHENDEN FILTERKONSTANTEN MULTIPLIZIERT
0028 00D5                UND DAS ERGEBNIS IM ACCA AUFADDIERT
0028 00D5
0028 00D5  */
0028 00D5  LOOP:         OP MOV @B,MEM  ;/* X(I) IN ACCB LADEN */
0029 00D6                OP MOV @MEM,TR  /* X(I-1) RUCKSPEICHERN */
0029 00D6                   M4          ;/* ZEIGER AUF X(127-I) SETZEN */
0030 00D7                OP MOV @TR,B    /* X(I) IN TR ZWISCHENSPEICHERN */
0030 00D7                   ADD ACCB,RAM /* X(127-I)  ADDIEREN */
0030 00D7                   DPINC        ;/* ZEIGER AUF X(127-I+1) SETZEN */
0031 00D8                OP MOV @KLR,B  ;/* MULTIPLIKATIONSWERTE LADEN */
0032 00D9                OP MOV @B,MEM   /* X(127-I+1) --) ACCB */
0032 00D9                   ADD ACCA,M    /* PRODUKT AUFADDIEREN */
0032 00D9                   DPDEC         /* X(127-I) ADRESSIEREN */
0032 00D9                   RPDEC        ;/* NAECHSTEN FILTERKOEFFIZIENTEN ADRESSIEREN */
0033 00DA                OP MOV @MEM,B   /* X(127-I) MIT X(127-I+1) UEBERSCHREIBEN */
0033 00DA                   DPINC.        /* X(I+1) ADRESSIEREN */
0033 00DA                   M4           ;
0034 00DB                JDPLO $+2       ;/* WENN DPL = 0 ENDE DER SCHLEIFE */
0035 00DC                JMP LOOP        ;
0036 00DD                OP M5            /* ZEIGER MODIFIZIEREN */
0036 00DD                   RET          ;/* RETURN INS HAUPTPROGRAMM */
0037 00DE  EOF;
```

Bild 3. Ein 128gradiges FIR-Filter läßt sich mit diesem Programm realiseren

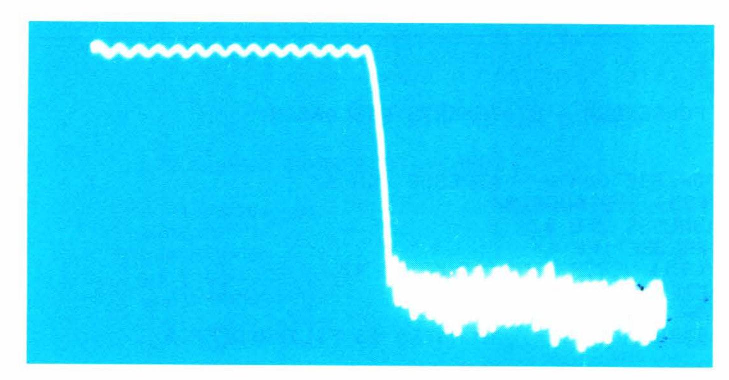


Bild 4. Frequenzgang eines linearphasigen Tiefpasses 128. Grades, der mit dem Signalprozessor µPD 7720 aufgebaut wurde

Speicherbelegung für die in Bild 2 dargestellte Filterstruktur

DP_L / DP_H	0000	0001	0010	...	...	1110	1111
100	X_0	X_1	X_2	...	...	X_{14}	X_{15}
101	X_{16}	X_{17}	X_{18}		.	X_{30}	X_{31}
110	X_{32}	X_{33}	X_{34}		.	X_{46}	X_{47}
111	X_{48}	X_{49}	X_{50}		.	X_{62}	X_{63}
000	X_{127}	X_{126}	X_{125}		.	X_{113}	X_{112}
001	X_{111}	X_{110}	X_{109}		.	X_{97}	X_{96}
010	X_{95}	X_{94}	X_{93}		.	X_{81}	X_{80}
011	X_{79}	X_{78}	X_{77}		.	X_{65}	X_{64}

sors müßte das Programm (*Bild 3*) zu verstehen sein. Zur Eingabe der Abtastwerte wird ein Codec des Typs 2910 verwendet, dessen nichtlineare Kennlinie (µ-law) bei der Eingabe über eine ROM-Tabelle, bei der Ausgabe über ein Unterprogramm (LINMIU) entzerrt wird [2]. Das Startsignal zum Umsetzen eines Abtastwertes beim Codec ist mit dem Interrupteingang des Signalprozessors verbunden, so daß auch das Programm startet. Am Ende des Programms wird der jetzt digital vorliegende Abtastwert in das Register TR geladen. Die Laufzeit des Programms garantiert, daß die Analog/Digital-Umsetzung abgeschlossen ist.

Will man ein linearphasiges FIR-Filter vom Grad 127 realisieren, müssen im Programm zwei Befehle geändert werden, und ein Befehl ist hinzuzufügen. Darauf wird in den Kommentaren hingewiesen. Diese geringfügige Änderung ergibt sich nur deshalb, weil die Speicherplatzorganisation wie bei N = 128 beibehalten wurde, es ist lediglich dafür gesorgt, daß am Ende des Programms X_{63} nach X_{65} umgespeichert wird und X_{64} immer den Wert 0 hat.

Das Programm läßt sich auch für Filtergrade N kleiner 128 verwenden. Auch in diesem Fall sind einige Teile des Programms wegzulassen, worauf bei den Kommentaren hingewiesen wird.

Linearphasiger Tiefpaß 128. Grades

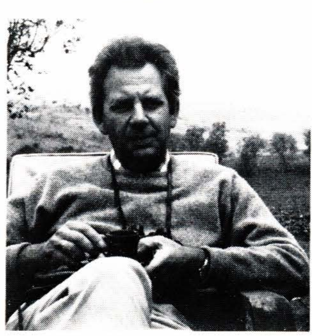
Mit Hilfe des „Remez-Exchange-Algorithm"-Programms [3], das auf der Rechenanlage VAX 11/780 der Fachhochschule Furtwangen implementiert ist, wurde ein linearphasiger Tiefpaß 128. Grades mit folgenden Daten dimensioniert:

Abtastfrequenz	8	kHz
Durchlaßgrenzfrequenz	2	kHz
Sperrfrequenz	2,08	kHz
Dämpfung im Durchlaßbereich	1,7	dB
Dämpfung im Sperrbereich	40	dB

Bild 4 zeigt den gemessenen Frequenzgang des Tiefpasses. Die Schwankungen im Sperrbereich (<2 dB) sind auf die Genauigkeit der Koeffizienten (13 Bit) zurückzuführen.



Dipl.-Ing. (FH) Thomas Dischinger ist in Donaueschingen geboren. Er studierte an der Fachhochschule Furtwangen Elektronik. Erste Erfahrungen in der digitalen Signalverarbeitung sammelte er im zweiten Industriesemester bei Hewlett-Packard. Im Rahmen seiner Diplomarbeit entstand dieser Artikel. Momentan setzt er sein Studium an der TU München fort.
Hobbys: Tennis und Heimwerken



Prof. Dr.-Ing. Horst Nielinger ist gebürtiger Westfale. Er studierte in Karlsruhe und trat 1960 bei AEG-Telefunken ein. Nach seiner Promotion 1967 in München übernahm er die Leitung eines Senderlabors in Ulm. Seit 1971 ist er Dozent für Nachrichtentechnik an der Fachhochschule Furtwangen.
Liebhabereien: Familienwanderungen im Schwarzwald, Barockmusik auf Blockflöte

Literatur

[1] Signal Processing Interface (SPI) µPD 7720. Product Description. NEC Europe, Juli 1981.
[2] µPD 7720 Workshop. NEC Europe, 1982.
[3] Programs for Digital Signal Processing. IEEE Press, New York 1979.

Dipl.-Ing. Gerhard Kratz, Dipl.-Ing. Walter Schumacher

Signalprozessor optimal an Mikrocomputer angekoppelt

Moderne Signalprozessoren – ursprünglich für den Bereich der Nachrichtentechnik konzipiert – werden aufgrund ihrer Leistungsfähigkeit auch für Echtzeitanwendungen im Bereich der Meß- und Regeltechnik (z. B. Antriebstechnik, Netzschutz) interessant. Hierbei wird der Signalprozessor nur in den seltensten Fällen die gesamte Prozeß-Beobachtung bzw. -Regelung allein durchführen können; er muß in ein komplexes System integriert werden. Am Beispiel des Bausteins 7720 werden im folgenden mögliche Konfigurationen und Kopplungsarten des Signalprozessors mit seiner „Umwelt" vorgestellt.

Mit der Fortentwicklung der Mikroelektronik rücken komplexe Regelverfahren und Identifizierungsalgorithmen, die bisher nur theoretisch untersucht werden konnten, in den Bereich des praktischen Einsatzes. Denkbare Anwendungsbereiche sind:
- dynamisch hochwertige Regelung von Drehstrommaschinen,
- schnelle Netzzeigeridentifizierung und Fehlerartanalyse nach Störungen im Energieversorgungsnetz (Netzschutz),
- Koordinatentransformationen bei Industrierobotern,
- schnelle Fourier-Transformation zur Oberwellenanalyse und Regelung bei Stromrichtern (Traktionsantriebe).

Derartige schnelle Echtzeitanwendungen, in denen pro Abtastperiode viele Rechenoperationen abgearbeitet werden, erfordern Rechner, die erheblich leistungsfähiger sind als die heute üblichen 16-Bit-Mikroprozessoren. In diesen Fällen mußten bisher „maßgeschneiderte" Rechenwerke vom Umfang mehrerer Europakarten aus Bit-Slice-Prozessoren und Hardwaremultiplizierern eingesetzt werden.

Von verschiedenen Halbleiterherstellern wurden in letzter Zeit spezielle Einchip-Signalprozessoren auf dem Markt eingeführt, die diese Anforderungen mit einem Minimum an Hardwareaufwand und gleichzeitig hoher Flexibilität erfüllen [1]. Durch interne 16-Bit-Architektur, Parallelstruktur, mehrere Datenbusse sowie Hardwaremultiplizierer auf dem Chip, ergibt sich bei den genannten Anwendungen ein Vielfaches der Verarbeitungsleistung gegenüber einem 16-Bit-Mikroprozessor. Durch den internen Programm- und Datenspeicher ist der Aufbau eines kompakten Systems möglich.

Obwohl ein Einchip-Signalprozessor auch im Alleinbetrieb einsetzbar ist, wird im allgemeinen Fall ein übergeordneter Rechner zur Anpassung von Filterkoeffi-

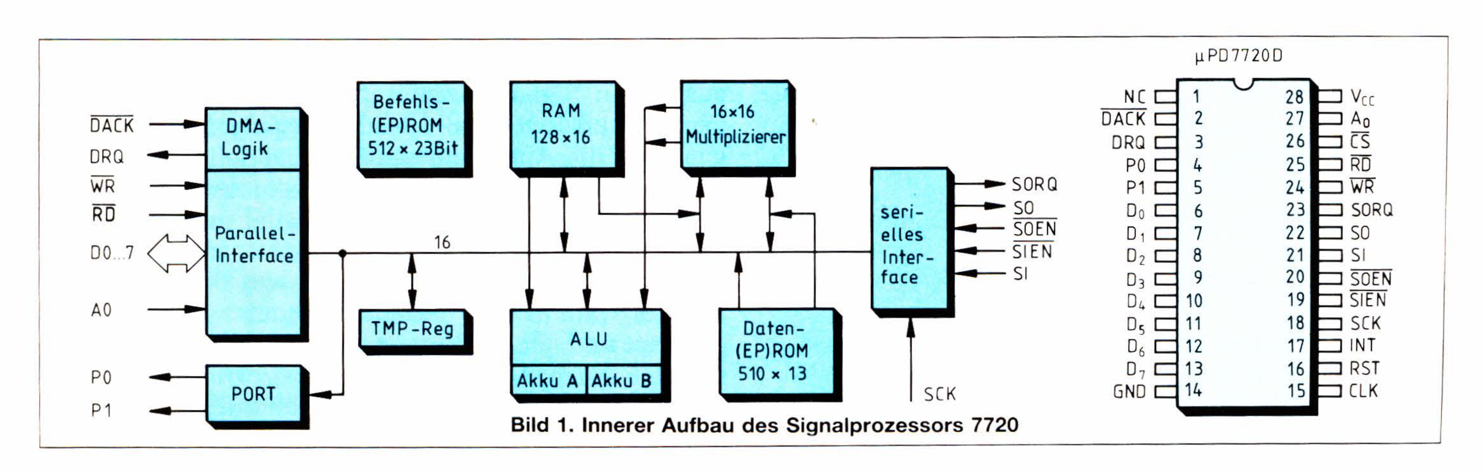


Bild 1. Innerer Aufbau des Signalprozessors 7720

zienten, Reglerparametern, Vorgabe von Sollwerten, zur Weiterverarbeitung der Ergebnisse oder als Bedienerschnittstelle vorhanden sein. Zur Kopplung an den Prozeß ist in den meisten Fällen eine Analog-/Digital-Schnittstelle erforderlich, die sowohl am Mikrorechner als auch am Signalprozessor angeschlossen sein kann. Die Verbindung beider Prozessoren stellt eine Koppelschnittstelle her, deren Umfang von der zu übertragenden Datenrate abhängt.

1 Der Signalprozessor NEC 7720

Im folgenden soll die Problematik anhand des Bausteins NEC 7720 erläutert werden, der praktisch als erster leistungsfähiger Einchip-Signalprozessor verfügbar war. Die typischen Merkmale eines „Signalprozessors" werden an der in *Bild 1* stark vereinfacht dargestellten inneren Struktur deutlich. Befehls- und Datenfestwertspeicher, Schreib-/Lesespeicher, 16×16-Bit-Multiplizierer, arithmetisch/logische Einheit sowie serielle und parallele Schnittstelle sind auf dem Chip durch mehrere Busse miteinander verbunden. Diese interne Parallelstruktur ermöglicht das gleichzeitige Abarbeiten von mehreren verschiedenen Operationen in einem Befehlszyklus (250 ns). Folgende drei Operationen können beispielsweise simultan ablaufen:
- Laden von zwei 16-Bit-Faktoren in den Multiplizierer, Multiplikation.
- Verschieben eines Datenwortes vom RAM zur ALU (Operand 1).
- Arithmetische Verknüpfung von Operand 1 mit dem Inhalt eines Akkumulators.

Das Beispiel läßt bereits vermuten, daß eine alle Möglichkeiten ausnutzende Programmierung dieses Bausteins aufwendiger ist als die Assemblerprogrammierung von Mikrorechnern. Dafür ergibt sich andererseits bei praktisch ausgeführten Programmen zur digitalen Signalverarbeitung z. B. gegenüber einem 8086 mit 8-MHz-Takt eine Laufzeitverbesserung um den Faktor 5...20. Für die Ankopplung des Signalprozessors an seine Umgebung bestehen verschiedene Möglichkeiten, abhängig von den Anforderungen an das Gesamtsystem.

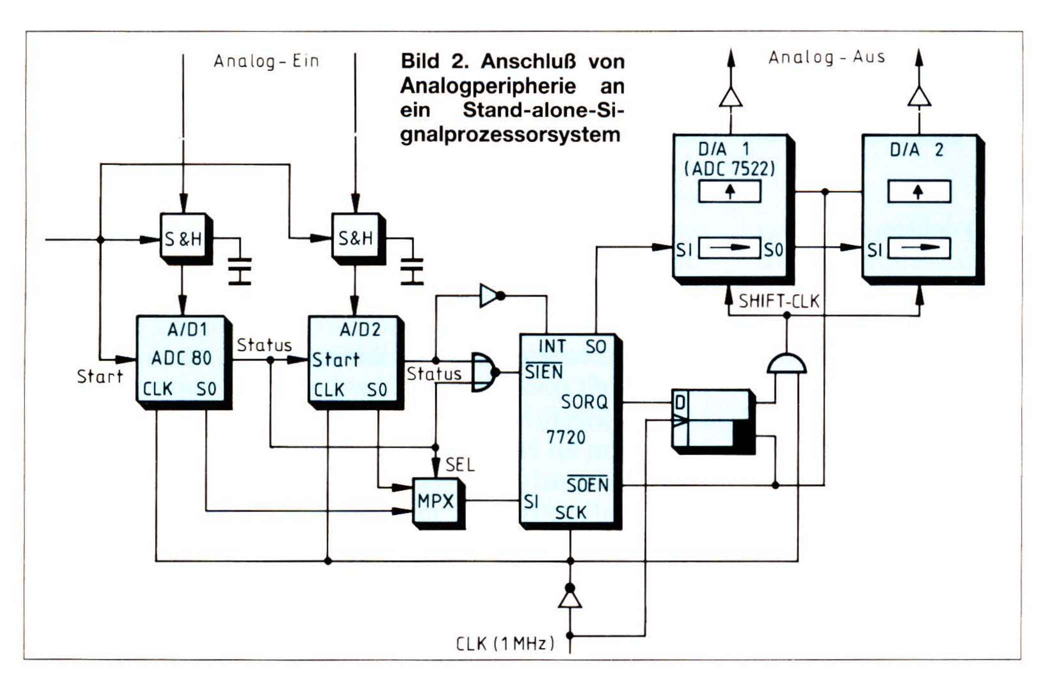


Bild 2. Anschluß von Analogperipherie an ein Stand-alone-Signalprozessorsystem

2 Schnittstellen

Bei der Regelung bzw. Identifizierung eines technischen Prozesses mit einem Rechner sind pro Abtastintervall eine Reihe von Abtastwerten in den Rechner hinein- und eine Reihe von Stell- bzw. Ergebnisgrößen herauszutransferieren.

Bei einem schnellen oder komplexen Prozeß, für den ein Signalprozessor eingesetzt werden soll, wird die Tatsache bedeutsam, daß dieser Baustein in der Abarbeitung arithmetischer Befehle sehr schnell ist, der Datentransfer hingegen wegen der relativ langsamen seriellen Schnittstellen des Signalprozessors bzw. der begrenzten Arbeitsgeschwindigkeit der am Transfer beteiligten Systembausteine (z. B. angekoppelter Mikrorechner) verzögert wird.

Ein Signalprozessor kann in verschiedenen Systemkonfigurationen betrieben werden. Den einfachsten Fall bildet der Alleinbetrieb (*Stand-alone*). Hier bietet sich beim 7720 die Verwendung der seriellen Schnittstellen an. In Ein- und Ausgaberichtung steht je ein Kanal mit einer Datenrate von 2 MBit/s zur Verfügung. Es lassen sich mit geringem Hardwareaufwand D/A-Umsetzer mit serieller Schnittstelle oder A/D-Umsetzer nach dem allseits bekannten Verfahren der sukzessiven Approximation anschließen. *Bild 2* zeigt die schematische Darstellung eines Systems aus einem Signalprozessor mit zwei 8-Bit-Analogeingängen und zwei 8-Bit-Analogausgängen. Ein Taktsignal, das die Abtastrate des Systems definiert, versetzt die Abtasthalteverstärker in den Haltezustand und startet den ersten A/D-Umsetzer. Parallel zur Umsetzung gelangen die Datenbits seriell (MSB zuerst) an den seriellen Eingang des Signalprozessors. Mit der Fertigmeldung (Status) des ersten Umsetzers wird der zweite gestartet und der Multiplexer auf diesen Baustein umgeschaltet. Nach Abschluß der zweiten Umsetzung wird ein Interrupt für den Signalprozessor ausgelöst, um die Verarbeitung der Werte zu veranlassen. Sobald die Aus-

gabewerte des Signalprozessors vorliegen, fordert er über die Leitung SORQ eine serielle Ausgabe an. Synchronisiert mit dem Ausgabetakt wird auf der Leitung SOEN die serielle Ausgabe freigegeben. Die verwendeten 8-Bit-D/A-Umsetzer besitzen integrierte Schieberegister, die in Reihenschaltung vom seriellen Ausgang SO gespeist werden. Am Ende des Schiebevorgangs (16 Bit) werden die Daten mit der Rückflanke von SOEN gleichzeitig in die Halteregister der beiden D/A-Umsetzer übernommen.

Die maximale Datenrate, die mit den seriellen Schnittstellen möglich ist, beträgt bei 16-Bit-Umsetzern etwa 125 kHz. In dieser Konfiguration lassen sich bereits komplexe digitale Filter, auch nichtlinearer Art, oder Regelalgorithmen (PID-Regler mit variabler Begrenzung) für hochdynamische Regelstrecken realisieren.

Reicht im Stand-alone-Betrieb die Leistungsfähigkeit der seriellen Schnittstellen nicht aus, weil in kurzer Zeit eine größere Anzahl von Werten zu transferieren ist, so bietet sich eine Erweiterung des Prozessors um eine aktive Busschnittstelle an. Die frei programmierbaren Portleitungen P0 und P1 veranlassen einen PAL-Baustein, die Adresse aus dem Prozessor zu lesen und zwei Buszyklen für die Übertragung der 16-Bit-Daten vorzusehen. Die Richtung des Transfers wird durch das Bit R/W gesteuert, das zusammen mit der Adresse übergeben wird. Mit dieser Lösung lassen sich 16-Bit-Daten in etwa 2,5 µs (16-Bit-Adreßraum) übertragen (*Bild 3*).

Sollen die Parameter des Filteralgorithmus im Signalprozessor von außen einstellbar sein oder wird eine Weiterverarbeitung von Ergebnissen des Signalprozessors mit niedriger Datenrate verlangt, so ist hierfür ein zusätzlicher Prozessor erforderlich, z. B. ein 8-Bit-(Einchip-)Mikroprozessor. Damit wird der Signalprozessor zu einem intelligenten Peripheriebaustein. Die Ankopplung an einen 8-Bit-Mikroprozessor ist im allgemeinen ohne besonderes Interface möglich. Die Datenraten des Transfers sind nicht durch den Signalprozessor, sondern durch die Verarbeitungsleistung und Busbandbreite des Mikroprozessors begrenzt (Z80A etwa 5 µs pro 16-Bit-Wort).

Wird der Signalprozessor innerhalb eines komplexen Systems zur Übernahme sehr rechenintensiver Teilaufgaben eingesetzt und ist die Weiterverarbeitung seiner Ergebnisse in Echtzeit erforderlich, ist es sinnvoll, ihn mit einem leistungsfähigen 16-Bit-Prozessor zu koppeln. Dabei stehen drei Möglichkeiten zur Wahl:
- direkte Ankopplung über 8-Bit-Schnittstelle,
- Verwendung eines DMA-Bausteins,
- Zwischenschaltung eines 8-/16-Bit-Multiplexers/-Demultiplexers.

Der erste Fall entspricht der Ankopplung eines 8-Bit-Mikroprozessors und ist mit dem geringsten schaltungstechnischen Aufwand verbunden, es ergibt sich jedoch keine volle Ausnutzung der Busbandbreite des angekoppelten 16-Bit-Prozessors.

Läßt das System den DMA-Betrieb zu, so sollte diese Alternative gewählt werden, da damit die maximale

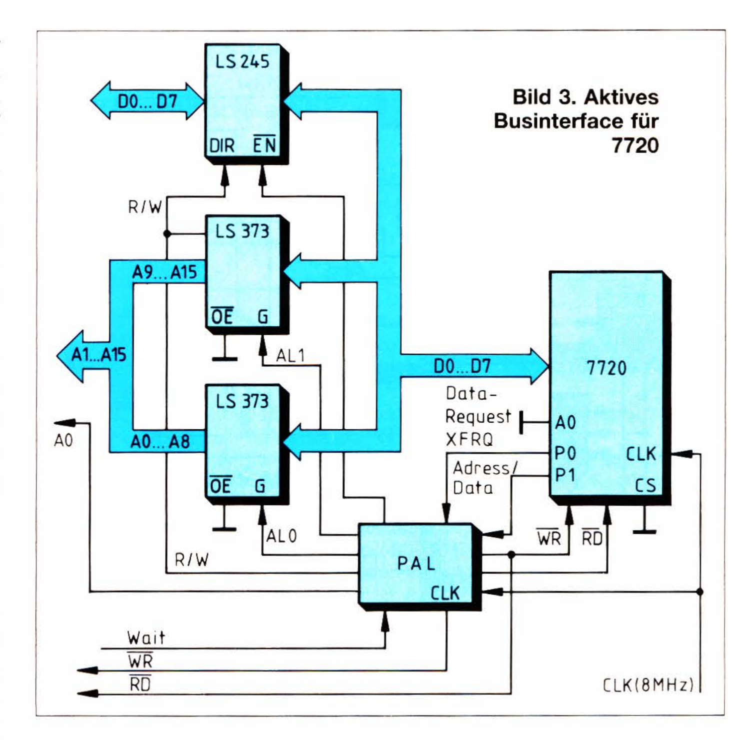


Bild 3. Aktives Businterface für 7720

Datenrate erzielbar ist (z. B. 8237: 1,25 µs, 8089: 2,4 µs; jeweils pro 16-Bit-Wort). Eine sehr elegante Lösung ergibt sich unter Verwendung des E/A-Prozessors 8089 zur Ankopplung des 7720 an ein System mit der CPU 8086 (*Bild 4*). Jeweils einer der beiden DMA-Kanäle des 8089 arbeitet in Ein- bzw. Ausgaberichtung zum 7720. Der Signalprozessor wird in die Betriebsart DMA-Mode versetzt. Der Transferanstoß erfolgt über die DMA-Anforderung des 7720 (DRQ); P0 wählt den Kanal und damit die Richtung des Transfers; die Beendigung des

Transferzeiten mit verschiedenen Prozessoren/Hardwarekonfigurationen

Konfiguration	Transferzeit pro 16-Bit-Wort (1)	begrenzt durch
Stand-alone mit serieller Schnittst. (2)	7,7 µs	serielle Schnittst. 7720
Stand-alone mit Parallelschnittst. (3)	1,0 µs	Parallel-Schnittst. 7720
7720 mit 8-Bit-M. a) 8048 (6 MHz) b) Z80A (4 MHz)	 22,5 µs 5,0 µs	 8048 Z80A
7720 mit 16-Bit-M. a) 8088 b) 8086 direkt c) 8086-Interf. (4)	 4,25 µs 3,75 µs 2,5 µs	 8088 8086-2 8086-2
7720 mit DMA-Baust. a) 8089 (5 MHz) (5) b) 8237	 2,4 µs 1,25 µs	 8089 8237

Anmerkungen: (1) mit Mikrorechner: Speichertransfer, (2) vgl. Bild 2, (3) ohne Adreßausgabe, vgl. Bild 3, (4) vgl. Bild 5, (5) vgl. Bild 4

Transfers erfolgt über P1/EXT oder durch Zählung der übertragenen Bytes im 8089.

Bei Systemen ohne DMA-Möglichkeit stellt eine Schaltung nach *Bild 5* einen guten Kompromiß dar. Der 16-Bit-Bus (z. B. des 8086) ist über je zwei 8-Bit-Register für beide Datenrichtungen mit dem 8-Bit-Bus des Signalprozessors verbunden.

Ein Schaltwerk, das als Eingangssignale die Bussteuersignale des 16-Bit-Prozessors und die vom Programm des Signalprozessors steuerbare Ausgabeleitung P0 verarbeitet, steuert den Multiplex-/Demultiplex-Betrieb der 8-/16-Bit-Daten. Jeder Buszyklus des 16-Bit-Prozessors wird durch das Steuerwerk in zwei Buszyklen des Signalprozessors umgesetzt.

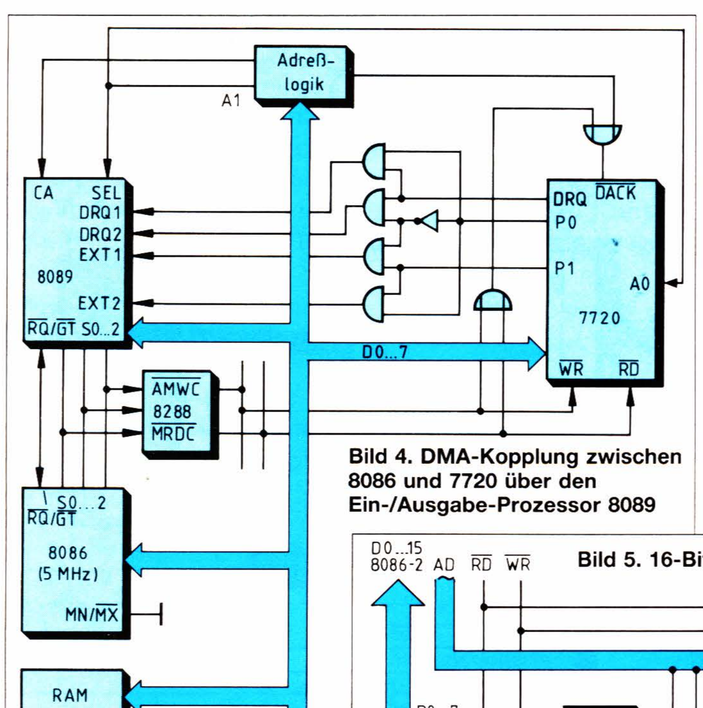


Bild 4. DMA-Kopplung zwischen 8086 und 7720 über den Ein-/Ausgabe-Prozessor 8089

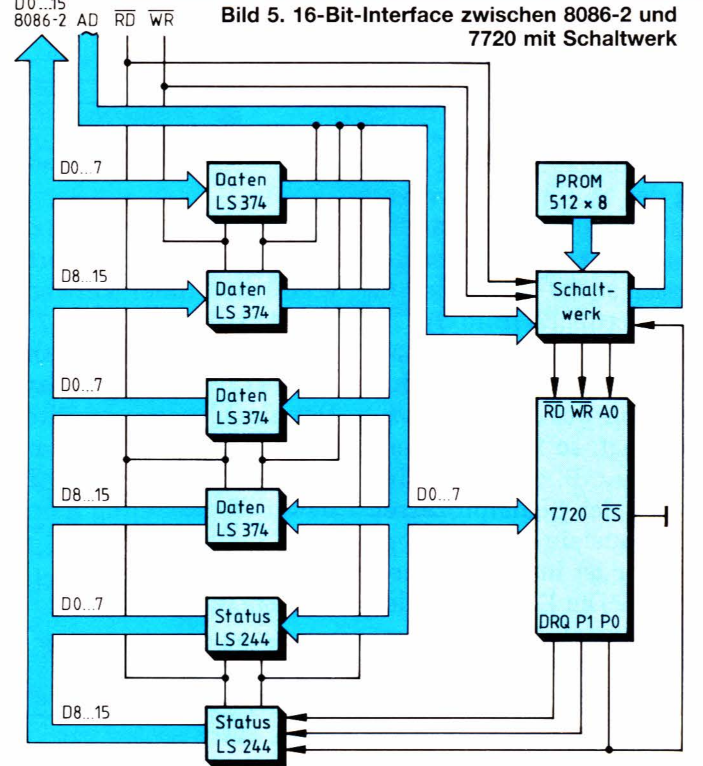
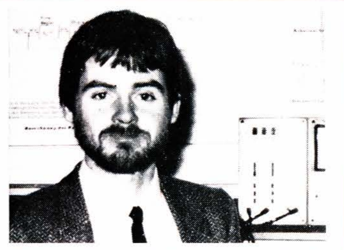


Bild 5. 16-Bit-Interface zwischen 8086-2 und 7720 mit Schaltwerk



Dipl.-Ing. Walter Schumacher wurde in Hamburg geboren, studierte an der TU Braunschweig Elektrotechnik mit den Schwerpunkten Meß- und Regelungstechnik/Datenverarbeitung. Seit 1979 ist er wissenschaftlicher Mitarbeiter am Institut für Regelungstechnik der TU Braunschweig; Arbeitsgebiet: mikrorechnergeregelte Drehstrom-Stellantriebe.
Hobbys: Mikrorechner, Fotografie



Dipl.-Ing. Gerhard Kratz, geboren in Lübeck, Studium der Elektrotechnik mit Schwerpunkt Regelungstechnik an der TU Braunschweig. Dort seit 1979 wissenschaftlicher Mitarbeiter am Institut für Regelungstechnik, Arbeitsgebiet: schnelle Zustandsidentifizierung im Energieversorgungsnetz mit Mikrorechnern.
Hobbys: Fotografie, Wassersport, Skifahren

Bei der Kopplung eines 7720 über ein derartiges Interface an einen am Institut für Regelungstechnik der TU Braunschweig entwickelten Einkarten-Mikrocomputer mit der CPU 8086-2 (8 MHz) ergaben sich Transferzeiten von 2,5 µs pro 16-Bit-Wort.

3 Anwendungsbeispiele

Standard-Mikrocomputer werden schon für viele Steuer- und Regelaufgaben bei geringen bis mittleren Anforderungen an die Verarbeitungsgeschwindigkeit industriell eingesetzt. Hingegen stößt man bei Anwendungen, die eine hohe Abtastrate verlangen, schnell an die Grenzen ihrer Leistungsfähigkeit, die auch durch erhöhten Programmieraufwand (Beschränkung auf Integerrechnung, Assemblerprogrammierung) nicht überwunden werden können. Als Beispiel seien hier zwei Forschungsprojekte aus dem Institut für Regelungstechnik der TU Braunschweig genannt:

- die Lage-Regelung einer Asynchronmaschine in Feldkoordinaten,
- die schnelle Netzzeigererkennung im Drehstromversorgungsnetz unter Verwendung von diskreter Fouriertransformation und Regressionsfiltern (selektiver Netzschutz).

In beiden Fällen sind aufwendige Rechenalgorithmen erforderlich. Bei diesen Anwendungen wurde jeweils ein Signalprozessor über ein Interface gemäß *Bild 5* an den Prozessor 8086-2 gekoppelt.

Bei der Regelung der Asynchronmaschine sind pro Abtastschritt 20 Multiplikationen mit 16 Bit erforderlich. Ein 8086-2 (8 MHz) benötigt für die Abarbeitung des gesamten Regelalgorithmus 800 µs. Um die Rechnung zu beschleunigen, wurden die gesamten Regelprogramme für die Asynchronmaschine

auf den Signalprozessor übertragen. Sie füllen den Programmspeicher zu etwa 85 %. Der Datenspeicher wird von den Variablen der Regelung zu etwa 60 % belegt. Nach Einsatz des Signalprozessors konnte die Abtastrate auf 100 µs gesenkt werden, so daß der Betriebsbereich der Regelung auf schnellaufende Servomotoren mit Ständerfrequenzen bis zu 500 Hz erweitert wird. *Bild 6* zeigt gemessene zeitliche Verläufe von Drehzahl, Drehmoment und Ständerströmen während eines Reversiervorgangs bei 1000 U/min einer leerlaufenden 1,5-kW-Kurzschlußläufer-Asynchronmaschine an einem 8-kVA-Transistorwechselrichter. Er ist in weniger als einer Schwingung der Ständerfrequenz abgeschlossen.

Im zweiten Anwendungsfall arbeitet der Signalprozessor als Coprozessor neben dem 8086-2 in einem Rechner für den Netzschutz. Hierbei besteht das zentrale Problem in der schnellen Identifizierung der Netzzeiger im Störungsfall (Kurzschluß). Der Signalprozessor übernimmt hierbei die Berechnung

- der drei Stromzeiger mit diskreter Fouriertransformation (DFT) oder mit Regressionsfiltern (bis 20. Ordnung),
- die Transformation der Zeiger in die „symmetrischen Komponenten" (*Bild 7*).

Das System arbeitet mit einer Abtastrate von 1600 Hz. Innerhalb eines Abtastintervalls von 625 µs werden im Programm des Signalprozessors unter anderem 150 Multiplikationen (entsprechend 240 000 Multiplikationen/s) durchgeführt. Fünf 16-Bit-Worte müssen pro Abtastperiode in den Signalprozessor hinein- und bis zu sieben Werte müssen heraustransferiert werden. Eine blockweise Datenübergabe zwischen den beiden Prozessoren ist erwünscht, um die Programme weitgehend zu entkoppeln; die entstehende Transferzeit von 30 µs macht bereits etwa 5 % der Rechenzeit aus. Dies unterstreicht die Bedeutung der Auswahl der optimalen Interfaceschaltung zwischen Mikrorechner und Signalprozessor.

Es handelt sich um eine Mitteilung aus dem Institut für Regelungstechnik der TU Braunschweig.

Literatur

[1] ELEKTRONIK 1982, H. 21, Sonderteil Signalprozessoren.
[2] µPD 7720 Signalprocessing Interface Product Description. Fa. NEC.
[3] The MCS86 User's Manual. Fa. Intel.
[4] 1982/83 Data Book. Fa. Zilog.
[5] *Rojek, P.*: Regelung eines asynchronen Stellantriebs mit Signalprozessor. Diplomarbeit, Inst. f. Regelungst., TU Braunschweig, 1982.
[6] *Ernst, J.*: Schnelle Nachführung des Flußmodells bei einem Mikrorechnergeregelten asynchronen Stellantrieb. Diplomarbeit, Inst. f. Regelungst., TU Braunschweig, 1982.
[7] *Lindemann, U.*: Netzzeigeridentifikation mit Signalprozessor. Diplomarbeit, Inst. f. Regelungst., TU Braunschweig, 1983.
[8] *Koblischke, R.*: Entwurf und Aufbau von Hauptrechner- und Analogschnittstelle für einen Signalprozessor. Studienarbeit, Inst. f. Regelungst., TU Braunschweig, 1983.
[9] *Schumacher, W., Leonhard, W.*: Transistor-Fed AC Servo Drive With Microprocessor Control. International Power Electronics Conference, Tokio, 1983.
[10] *Kratz, G., Reupke, J.*: Ein neues Verfahren für den digitalen Leitungsschutz in Hochspannungsnetzen. etz-Archiv. Erscheint 1984.

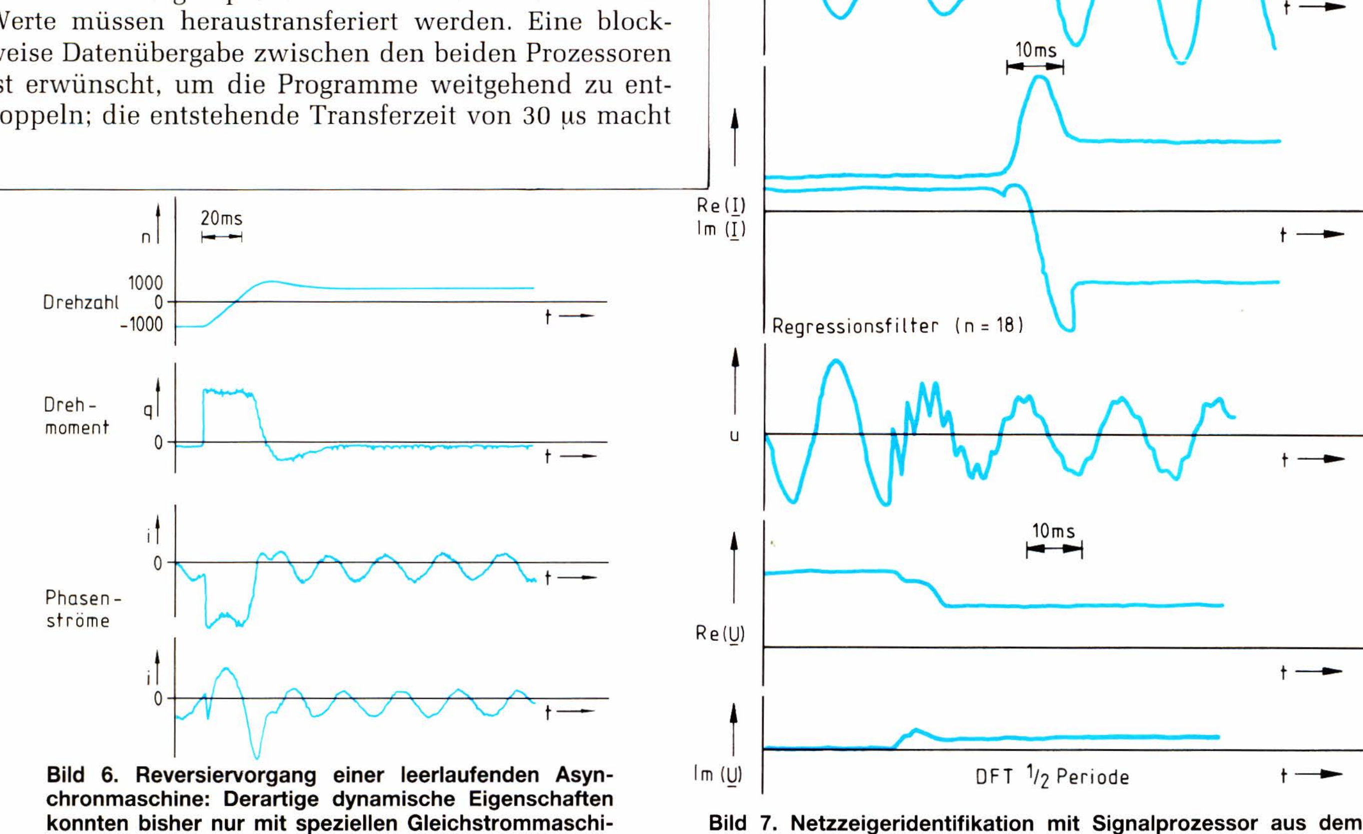


Bild 6. Reversiervorgang einer leerlaufenden Asynchronmaschine: Derartige dynamische Eigenschaften konnten bisher nur mit speziellen Gleichstrommaschinen erreicht werden

Bild 7. Netzzeigeridentifikation mit Signalprozessor aus dem Verlauf von Strom und Spannung

Dr.-Ing. Herbert Hanselmann

Tischrechner programmiert Signalprozessor als digitalen Mehrgrößenregler

Für anspruchsvolle Aufgaben der Regelung dynamischer Systeme werden zunehmend die neueren Methoden der Regelungstechnik eingesetzt, die bei voller Ausnutzung ihrer Möglichkeiten auf Regler höherer Ordnung mit mehreren Eingängen (Meßgrößen der Regelstrecke) und Ausgängen (Stellgrößen) führen [1, 2, 3]. Aber auch konventionelle Konzepte wie die Kaskadenregelung bei Gleichstromantrieben führen auf komplexere Systeme, wenn besondere regelungstechnische Maßnahmen wie Führungsgrößenfilterung oder Kompensation schwingender Teilsysteme ergriffen werden sollen. Der folgende Beitrag beschreibt ein Programmsystem für FORTRAN-Tischrechner, das zusammen mit dem preiswerten „2920 System Design Kit" die automatische Programmierung des Signalprozessors 2920 erlaubt.

Die Realisierung solcher Regler mit analogen Mitteln kann wegen der hohen Anzahl dynamischer Elemente und der gegebenenfalls starken Vernetzung untereinander zu erheblichen Problemen führen, die eine digitale Realisierung ratsam erscheinen lassen. Bei schnellen Regelstrecken, wie sie z. B. in der Elektromechanik und Elektrohydraulik vorkommen, müssen allerdings Tastraten bis in den kHz-Bereich erzielt werden. Hier sind selbst moderne 16-Bit-Mikroprozessoren schnell überfordert. Mit einem oder mehreren parallel- oder in Serie geschalteten Signalprozessoren des Typs 2920 (Intel) können die zeitaufwendigen Filter-/Regleroperationen zum Teil auf analoger Ebene vom Mikroprozessorsystem abgezogen werden [4, 5, 6]. Wo die 9-Bit-A/D-Umsetzung ausreicht, sind auch Regler allein auf der Basis von 2920-Bausteinen möglich.

Die Programmierung des 2920 für den beschriebenen Zweck sollte soweit wie möglich automatisch erfolgen. Zum einen, weil die Programmstruktur komplexer wird als für ein übliches digitales Filter, zum anderen, weil ein Regler in den seltensten Fällen vom grünen Tisch weg endgültig realisiert werden kann. Meist sind noch Korrekturen nötig, die dann auch möglichst schnell umsetzbar sein sollten. Mit dem im folgenden beschriebenen Programmierungssystem wird der Zeitaufwand für die Programmierung eines zahlenmäßig vorliegenden Mehrgrößenreglers bzw. -filters auf wenige Minuten reduziert.

Bild 1. Mehrgrößenregler/Filter in Zustandsform

1 Lineare digitale Mehrgrößenregler

Unabhängig von der Vielfalt der Methoden zur Auslegung und Berechnung digitaler Mehrgrößenregler ist das Endprodukt im allgemeinen entweder direkt das folgende Gleichungssystem (*Bild 1*), oder man kann es in diese Form bringen.

$$\underline{x}_k = \underline{A}\,\underline{x}_{k-1} + \underline{B}\,\underline{u}_{k-1} \quad (1)$$

oder

$$\underline{x}_{k+1} = \underline{A}\,\underline{x}_k + \underline{B}\,\underline{u}_k,$$
$$\underline{y}_k = \underline{C}\,\underline{x}_k + \underline{D}\,\underline{u}_k \quad (2)$$

Dabei sind die Elemente u_{ik} des Vektors (Anordnung der Elemente für Matrixschreibweise)

$$\underline{u}_k = \begin{bmatrix} u_{1k} \\ \cdot \\ \cdot \\ u_{pk} \end{bmatrix} \quad (3)$$

die Werte der p Eingangsgrößen des Reglers zum Zeitpunkt kT, mit T als konstante Tastzeit. Die Elemente des

Vektors $\underline{y}_k$ sind in entsprechender Weise die Werte der q Ausgangsgrößen des Reglers zum Zeitpunkt kT. Der Vektor $\underline{x}_k$ enthält die n internen zeitveränderlichen Größen des Reglers. Bei einem auf übliche Weise dargestellten digitalen Filter würden diese Größen die Ausgangsgrößen der z^{-1}-Blöcke repräsentieren [4, 5, 6]. Die Matrizen $\underline{A}$, $\underline{B}$, $\underline{C}$, $\underline{D}$ enthalten als Elemente die konstanten Koeffizienten des Reglers/Filters.

Die kompakte formalisierte Darstellung des Reglers/Filters in Zustandsform (Gleichung 1 und 2) ermöglicht eine systematische automatisierte Umsetzung in ein 2920-Programm. Liegt der Regler zunächst nicht in Zustandsform, sondern in Form von z-Übertragungsfunktionen vor, so muß eine dazu passende Zustandsrealisierung gefunden werden. Daß dies im konkreten Fall recht einfach möglich ist, soll das folgende kleine Beispiel zeigen.

Beispiel: PID-Regler mit Führungsgrößenfilterung (*Bild 2*).

Durch Einführung der sogenannten Zustandsgrößen $x_1 ... x_4$ an den Ausgängen der z^{-1}-Blöcke erhält man für den Zeitpunkt kT

$$\left.\begin{aligned} x_{1k} &= -\beta_0\, x_{2\,k-1} + K(\alpha_0 - \alpha_2\,\beta_0)\, f_{k-1} \\ x_{2k} &= x_{1\,k-1} - \beta_1\, x_{2\,k-1} + K(\alpha_1 - \alpha_2\,\beta_1)\, f_{k-1} \\ x_{3k} &= \lambda_1\, x_{2\,k-1} + \lambda_1\, K\, \alpha_2\, f_{k-1} - \lambda_1\, m_{k-1} \\ x_{4k} &= \lambda_2\, x_{2\,k-1} + \lambda_2\, K\, \alpha_2\, f_{k-1} - \lambda_2\, m_{k-1} + \lambda_3\, x_{4\,k-1} \end{aligned}\right\} \quad (4)$$

$$\begin{aligned} s_k = &- x_{3k} + (\lambda_1 + \lambda_0)\, x_{2k} \\ &+ (\lambda_1 + \lambda_0)\, K\, \alpha_2\, f_k - (\lambda_1 + \lambda_0)\, m_k + x_{4k} \end{aligned} \quad (5)$$

Mit (4) und (5) ist die Darstellung nach (1) und (2) im Prinzip gefunden, sie kann noch formal angepaßt werden mit

$$\underline{u}_k = \begin{pmatrix} f_k \\ m_k \end{pmatrix}$$

als dem Vektor der Regler-Eingangsgrößen, und

$$y_k = s_k$$

als der hier zu einem Skalar reduzierten Ausgangsgröße. Die Matrixschreibweise erhält man mit

$$\underline{A} = \begin{bmatrix} 0 & -\beta_0 & 0 & 0 \\ 1 & -\beta_1 & 0 & 0 \\ 0 & \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 & \lambda_3 \end{bmatrix}$$

$$\underline{B} = \begin{bmatrix} K\alpha_0 + K\alpha_2 & 0 \\ K\alpha_1 & 0 \\ \lambda_1 K\alpha_2 & -\lambda_1 \\ \lambda_2\, K\alpha_2 & -\lambda_2 \end{bmatrix}$$

$$\underline{C} = (0,\ \lambda_1 + \lambda_0,\ -1,\ 1),$$

$$\underline{D} = ((\lambda_1 + \lambda_0)\, K\alpha_2,\ -(\lambda_1 + \lambda_0)).$$

2 Struktur und Generierung des 2920-Programms

Von der Struktur eines Programms für ein auf übliche Weise realisiertes Digitalfilter [4] unterscheidet sich die hier benötigte Struktur, abgesehen von der Zustandsform, noch im folgenden:

a) Es sind im allgemeinen mehrere Ein- und Ausgänge vorhanden.

b) Es kommt darauf an, zwischen dem Abtasten der Eingangsgrößen und der Ausgabe der Ausgangsgrößen möglichst wenig Zeit verstreichen zu lassen, da diese Zeiten als Totzeiten im Regelkreis wirksam werden.

Um a) und b) gerecht zu werden, kann das 2920-Programm nicht einfach die Gleichungen (1) und (2) nacheinander abarbeiten. Vielmehr müssen die im

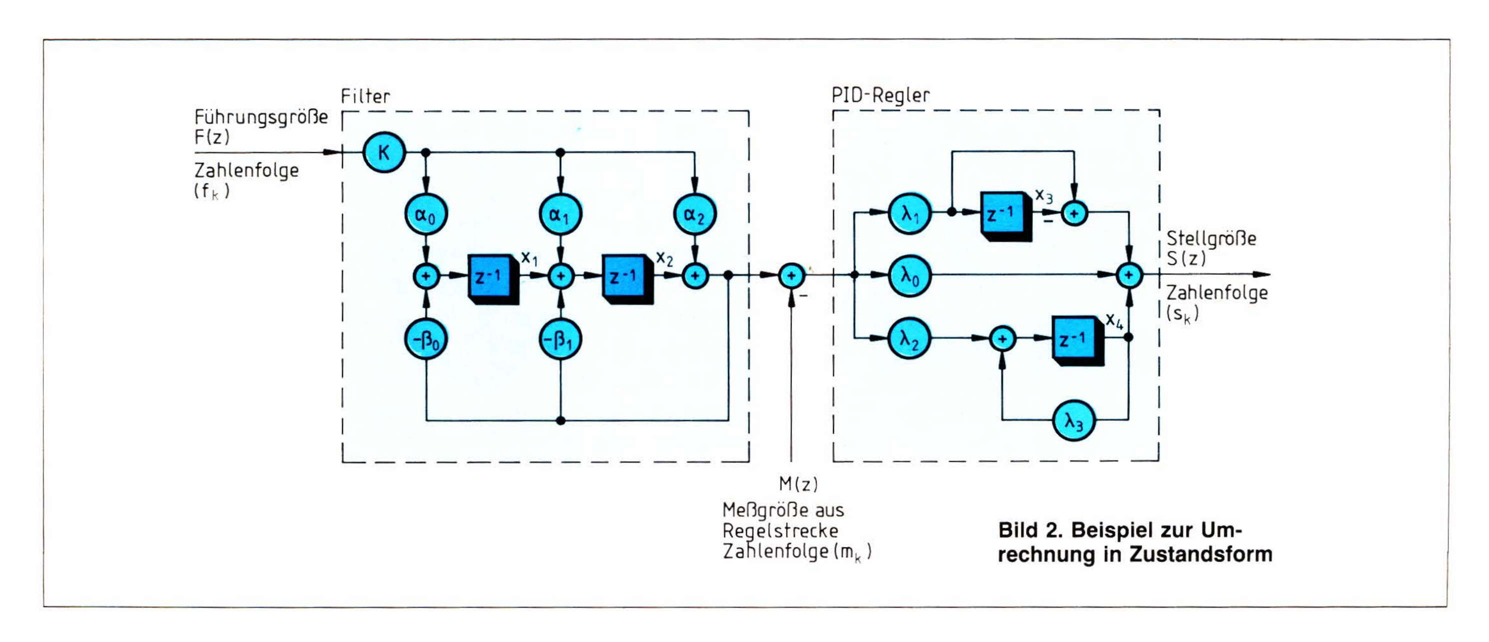


Bild 2. Beispiel zur Umrechnung in Zustandsform

Sinne von b) zeitkritischen Teile bevorzugt behandelt werden. Dies hat einige Konsequenzen für die Anordnung der Operationen im Programm wie auch für die Reihenfolge der Sourcecodeerzeugung für die einzelnen Teilaufgaben, die zur Bearbeitung der Reglergleichungen nötig sind.

Das Programm wird von einem in Mikrorechner-FORTRAN-80 geschriebenen Generatorprogramm als Sourcecode-Textfeld mit 192 Instruktionszeilen erzeugt. Darin sind eine Instruktionsnumerierung und ein automatisch erzeugter Kommentar mit enthalten. Das Generatorprogramm berücksichtigt die schaltungsspezifischen Daten (z. B. Anzahl der NOP-Instruktionen nach der Abtastung) und liest die numerischen Daten $\underline{A}$, $\underline{B}$, $\underline{C}$, $\underline{D}$ des Reglers von der Floppy-Disk. Die Umsetzung in Sourcecode erfolgt dann ohne weitere Eingriffe des Benutzers.

Der prinzipielle Ablauf beim Aufbau des Sourcecodes soll an Hand von *Bild 3* erläutert werden. Der spätere Programmablauf im 2920 entspricht der Pfeilrichtung von oben nach unten. Die Reihenfolge, in der die einzelnen Programmabschnitte im Tischrechner erzeugt werden, entspricht der Pfeilrichtung von links nach rechts. Es wird vereinfachend angenommen, daß zwei Eingänge und ein Ausgang vorliegen. Die Eintragung der Elemente a_{ij},... der Matrizen $\underline{A}$, $\underline{B}$, $\underline{C}$, $\underline{D}$ soll bedeuten, daß im zugehörigen Programmabschnitt die entsprechenden Matrixelemente gemäß (1) und (2) verarbeitet werden. Die Abarbeitung dieser Arithmetikoperationen läuft im 2920 parallel zu den Analogoperationen ab. Es kommt darauf an, diese Parallelität optimal auszunutzen, nicht nur wegen der Abtastzeit, die direkt proportional zur Anzahl belegter Instruktionen ist, sondern auch wegen des begrenzten Programmspeichers des Prozessors. Bei maximal 192 Instruktionen hat man wenig zu verschenken, wenn auch Regler etwa der Ordnung 5 oder höher mit mehreren Ein- und Ausgängen mit einem einzigen Chip realisiert werden sollen. Allein eine 9-Bit-Umsetzung benötigt bei 5 MHz Taktfrequenz und einem 300-pF-Sampling-Kondensator schon 34 Analoginstruktionen.

Beim Aufbau des Sourcecodes am Tischrechner werden vom Generatorprogramm zunächst die Analogbefehle für Abtasten und Umsetzen (IN/CVT-Sequenzen) am Programmanfang festgelegt (Bild 3). Daran anzuhängen sind die d-Sequenzen, d. h. die Verarbeitung der umgesetzten Abtastwerte u_i über die Matrix $\underline{D}$. Die Addition zum Term $\underline{C}x_k$ wird dabei gleich mit erledigt. Der Term $\underline{C}x_k$ liegt vom vorherigen Programmdurchlauf des 2920 schon fertig vor, er hängt nicht von den gerade gelesenen Eingangsgrößen u_i ab. Die Ausgabe der Ausgangsgröße y ist deshalb sofort nach Einrechnung des letzten D-Koeffizienten gemäß (2) möglich. Damit wird der kleinstmögliche Zeitverzug im Sinne von b) erreicht.

Die Bearbeitung der Gleichung (1), also die zu $\underline{A}$, $\underline{B}$, $\underline{C}$ gehörenden Arithmetikoperationen sind nicht zeitkritisch, sie werden deshalb nach der Festlegung der Ein-/Ausgabe- und d-Sequenzen lückenlos auf die noch freien Arithmetikinstruktionsfelder verteilt. So wird im Beispiel von Bild 3 mit der Verarbeitung der Elemente b_{i1} der 1. Spalte von $\underline{B}$ unter Verwendung von u_1 begonnen, sobald bei den zeitkritischen Operationen eine Lücke entsteht, also zwischen den beiden d-Sequenzen. Könnte die Verarbeitung der b_{i1} diese Lücke nicht ganz ausfüllen, würde die noch verbleibende Lücke mit Instruktionen zu den Elementen a_{ij} von $\underline{A}$ aufgefüllt usw. (*Bild 4*). Das Generatorprogramm muß in allen Fällen die entsprechende Aufteilung der Arithmetik automatisch vornehmen können, weil die optimale Lage der einzelnen Arithmetik-Sequenzen abhängig vom Zahlenmaterial stark variieren kann.

Soweit bis jetzt beschrieben, bleibt die erste IN/CVT-Sequenz für die Arithmetik noch ungenutzt. Um auch diese Instruktionen ausnutzen zu können, ermöglicht das Generatorprogramm eine Verschiebung von Instruktionen aus dem letzten Programmabschnitt in die erste IN/CVT-Sequenz. Diese Verschiebung wird dann nötig, wenn sonst mehr als 192 Instruktionen belegt werden müßten. In den anderen Fällen dient sie zur Minimierung der Programmlänge bzw. Abtastzeit.

Gegenüber Bild 3 enthält das reale 2920-Programm noch einige weitere Sequenzen, in denen Umspeicherungen (z. B. Inhalt des $\underline{x}_k$-Speichers $\rightarrow$ Inhalt des $\underline{x}_{k-1}$-Speichers im nächsten Durchlauf) und Vorbelegungen von Speicherplätzen erfolgen. In Bild 4 ist für einen Regler mit zwei Eingängen und einem Ausgang das generierte Programm angegeben, der Übersichtlichkeit wegen ohne die angesprochene Minimierung der Programmlänge.

Der Sourcecode von Bild 4 zeigt noch eine bisher nicht erwähnte Besonderheit. Soll z. B. der Koeffizient

Bild 3. Beispiel zur Aufteilung der 2920-Operationen

```
  0  SUB DAR,DAR,R00,IN0     Beginn Eingabe U1
  1                 ,IN0
  2                 ,IN0
  3                 ,IN0
  4                 ,IN0
  5                 ,NOP
  6                 ,NOP
  7                 ,CVTS
  8  ADD DAR,KM2,R00,CND6
  9                 ,NOP
 10                 ,NOP
 11                 ,CVT7
 12                 ,NOP
 13                 ,NOP
 14                 ,CVT6
 15                 ,NOP
 16                 ,NOP
 17                 ,CVT5
 18                 ,NOP
 19                 ,NOP
 20                 ,CVT4
 21                 ,NOP
 22                 ,NOP
 23                 ,CVT3
 24                 ,NOP
 25                 ,NOP
 26                 ,CVT2
 27                 ,NOP
 28                 ,NOP
 29                 ,CVT1
 30                 ,NOP
 31                 ,NOP
 32  LDA Y01,W01,R00,CVT0
 33  LDA U01,DAR,R00,NOP     Ende   Eingabe U1
 34  SUB DAR,DAR,R00,IN1     Beginn Eingabe U2
 35  ADD Y01,U01,R01,IN1     D(1,1)
 36  ADD Y01,U01,R02,IN1     D(1,1)
 37  ADD Y01,U01,R04,IN1     D(1,1)
 38  ADD Y01,U01,R05,IN1     D(1,1)
 39  SUB Y01,U01,R11,NOP     D(1,1)
 40  ADD Y01,U01,R13,NOP     D(1,1)
 41  LDA X01,KP0,R00,CVTS
 42  ADD DAR,KM2,R00,CND6
 43  LDA X02,KP0,R00,NOP
 44  LDA X03,KP0,R00,NOP
 45  ADD X01,U01,R11,CVT7    B(1,1)
 46  SUB X02,U01,R13,NOP     B(2,1)
 47  ADD X03,U01,R04,NOP     B(3,1)
 48  ADD X03,U01,R06,CVT6    B(3,1)
 49  ADD X03,U01,R07,NOP     B(3,1)
 50  ADD X03,U01,R10,NOP     B(3,1)
 51  ADD X03,U01,R12,CVT5    B(3,1)
 52  ADD X03,U01,R13,NOP     B(3,1)
 53  ADD X01,Z01,R00,NOP     A(1,1)
 54  SUB X01,Z01,R08,CVT4    A(1,1)
 55  ADD X01,Z01,R12,NOP     A(1,1)
 56  ADD X01,Z01,R13,NOP     A(1,1)
 57  SUB X02,Z01,R08,CVT3    A(2,1)
 58  SUB X02,Z01,R11,NOP     A(2,1)
 59  ADD X01,Z02,R09,NOP     A(1,2)
 60  ADD X01,Z02,R11,CVT2    A(1,2)
 61  ADD X01,Z02,R13,NOP     A(1,2)
 62  ADD X02,Z02,R00,NOP     A(2,2)
 63  SUB X02,Z02,R08,CVT1    A(2,2)
 64  ADD X02,Z02,R12,NOP     A(2,2)
 65  ADD X02,Z02,R13,NOP     A(2,2)
 66  ADD X03,Z03,R01,CVT0    A(3,3)
 67  LDA U02,DAR,R00,NOP     Ende   Eingabe U2
 68  SUB Y01,U02,L01,NOP     D(1,2)
 69  SUB Y01,U02,R02,NOP     D(1,2)
 70  SUB Y01,U02,R04,NOP     D(1,2)
 71  SUB Y01,U02,R06,NOP     D(1,2)
 72  ADD Y01,U02,R09,NOP     D(1,2)
 73  SUB Y01,U02,R11,NOP     D(1,2)
 74  LDA DAR,Y01,R00,NOP     Beginn Ausgabe Y1
 75  SUB X01,U02,R10,NOP     B(1,2)
 76  SUB X01,U02,R11,NOP     B(1,2)
 77  SUB X01,U02,R13,NOP     B(1,2)
 78  SUB X02,U02,R06,NOP     B(2,2)
 79  SUB X02,U02,R11,NOP     B(2,2)
 80  SUB X02,U02,R13,NOP     B(2,2)
 81  SUB X03,U02,R02,NOP     B(3,2)
 82  SUB X03,U02,R05,NOP     B(3,2)
 83  SUB X03,U02,R06,OUT0    B(3,2)
 84  SUB X03,U02,R08,OUT0    B(3,2)
 85  SUB X03,U02,R10,NOP     B(3,2)
 86  SUB X03,U02,R11,NOP     B(3,2)
 87  SUB X03,U02,R13,NOP     B(3,2)
 88  ADD X03,Z03,R02,NOP     A(3,3)
 89  ADD X03,Z03,R04,NOP     A(3,3)
 90  ADD X03,Z03,R05,NOP     A(3,3)
 91  ADD X03,Z03,R06,NOP     A(3,3)
 92  SUB X03,Z03,R09,NOP     A(3,3)
 93  ADD X03,Z03,R13,NOP     A(3,3)
 94  LDA W01,KP0,R00,NOP
 95  ADD W01,X01,R01,NOP     C(1,1)
 96  ADD W01,X01,R03,NOP     C(1,1)
 97  ADD W01,X01,R04,NOP     C(1,1)
 98  SUB W01,X01,R07,NOP     C(1,1)
 99  ADD W01,X01,R11,NOP     C(1,1)
100  ADD W01,X01,R12,NOP     C(1,1)
101  SUB W01,X02,R02,NOP     C(1,2)
102  SUB W01,X02,R04,NOP     C(1,2)
103  ADD W01,X02,R07,NOP     C(1,2)
104  SUB W01,X02,R11,NOP     C(1,2)
105  SUB W01,X02,R13,NOP     C(1,2)
106  SUB W01,X03,R00,NOP     C(1,3)
107  SUB W01,X03,R03,NOP     C(1,3)
108  SUB W01,X03,R04,NOP     C(1,3)
109  SUB W01,X03,R07,NOP     C(1,3)
110  SUB W01,X03,R08,NOP     C(1,3)
111  SUB W01,X03,R11,NOP     C(1,3)
112  SUB W01,X03,R12,NOP     C(1,3)
113  LDA Z01,X01,R00,NOP
114  LDA Z02,X02,R00,NOP
115  LDA Z03,X03,R00,NOP
```

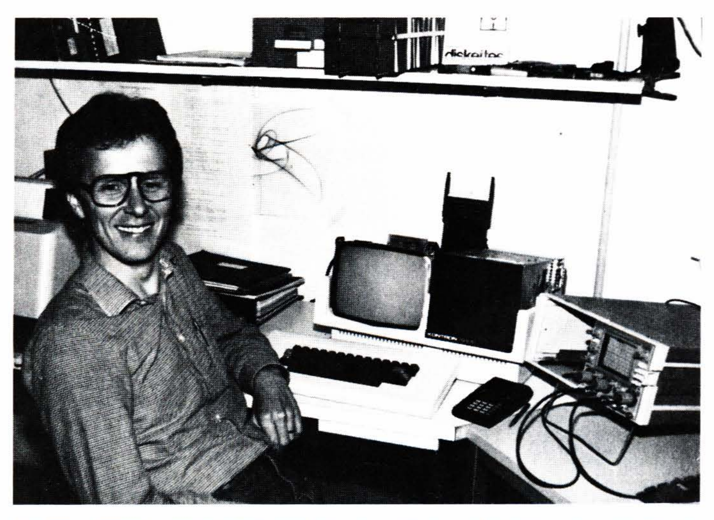
Bild 4. Beispiel für ein generiertes Programm

$0.111_B = 0.875_D$
realisiert werden, so ist dies mit drei Instruktionen in der Form (Rx bedeutet Rechtsshift um x Stellen)
ADD Zieladresse, Quelladresse, R01
ADD Zieladresse, Quelladresse R02
ADD Zieladresse, Quelladresse R03
aber auch mit nur zwei Instruktionen in der Form
ADD Zieladresse, Quelladresse, R00
SUB Zieladresse, Quelladresse R03
möglich, ohne den Wert des realisierten Koeffizienten zu ändern [4, 6]. Das Generatorprogramm nutzt diese Möglichkeit zur Verkürzung des Sourcecodes und damit zur Geschwindigkeitserhöhung aus. Dazu werden in der Binärdarstellung der Koeffizienten Ketten von drei oder mehr 1-Bits gesucht. Diese Ketten werden dann durch nur noch zwei Bits mit unterschiedlichen Vorzeichen (Instruktionen ADD und SUB) ersetzt. So kann z. B. der Koeffizient A(1,1) in Bild 4 durch vier Instruktionen realisiert werden gegenüber neun Instruktionen ohne die gemischte Verwendung von ADD und SUB.

3 Das Gesamtprogrammsystem

Das Programmsystem besteht derzeit aus:

a) dem beschriebenen Sourcecode-Generator,
b) einem Objekt- und Source-Transferprogramm vom Floppy zum „2920 System Design Kit" und zurück,
c) unterstützenden Programmen zur Skalierung und zum Vergleich des durch den 2920 realisierten digitalen Systems mit dem gewünschten, durch (1) und (2) repräsentierten System,
d) einem Assemblierprogramm.



Dr.-Ing. Herbert Hanselmann studierte an seinem Geburtsort Karlsruhe Elektrotechnik mit Schwerpunkt Regelungstechnik und wurde im Anschluß daran 1973 wissenschaftlicher Assistent beim dortigen Institut für Regelungs- und Steuerungssysteme bei Prof. Dr. O. Föllinger. Nach der Promotion im Jahre 1978 über ein Thema aus der Regler-Entwurfstheorie wechselte er zur Universität – Gesamthochschule – Paderborn in die Fachgruppe Automatisierungstechnik (Prof. Dr. J. Lückel), wo er sich mit Fragen der Auslegung und Anwendung schneller linearer, insbesondere digitaler Mehrgrößenregelsysteme beschäftigt.

Vorausgesetzt wird die Verfügbarkeit eines 64-K-Tischrechners (es wurde ein PSI80 der Fa. Kontron verwendet) mit FORTRAN-80-Programmiermöglichkeit und des „2920 System Design Kits SDK", der eine sehr preiswerte Möglichkeit zum Einstieg in die Anwendung des 2920 darstellt. Das erzeugte Programm kann über V24-Schnittstelle vom Tischrechner in den SDK geschrieben werden. Von dort aus wird das 2920-EPROM programmiert. Denkbar wäre auch, über ein kleines Treiberprogramm den Objektcode (6 × 4 Bit pro Instruktion) direkt in ein geeignetes EPROM-Programmiergerät zu schreiben. Der SDK könnte dann entfallen.

Merkwürdigerweise enthält der SDK zwar einen Assembler für Sourcecode, der über den vorhandenen, allerdings recht spartanischen SDK-Editor über Tastatur eingegeben wurde, Sourcecode über die Schnittstelle einzulesen und zu assemblieren ist aber nicht vorgesehen. Es ist deshalb nötig, für die Assemblierung selbst am Tischrechner zu sorgen. Damit besteht auch die Möglichkeit, den komfortableren Editor des Tischrechners zu verwenden. Eine nachträgliche Editierung ermöglicht Erweiterungen des generierten Programms, z. B. die Einfügung nichtlinearer Terme.

Die unter c) genannten Programme gestatten vor der EPROM-Programmierung eine Vorabprüfung, ob die Abweichungen durch die im 2920 begrenzte Koeffizienten-Wortlänge tolerierbar sind und ob Probleme bei der Aussteuerbarkeit (Overflow, Underflow) entstehen können. Gegebenenfalls ist eine Umskalierung vorzunehmen.

Abschließend sei vermerkt, daß der Regler von Bild 4 die quasi-analoge Regelung eines elektromechanischen Systems mit einer Abtastfrequenz von etwa 10 kHz bei 5 MHz Taktrate des 2920 erlaubte. Demgegenüber konnten mit einem Z80A/am9511-System in 32-Bit-Gleitkommarechnung (32-Bit-Festkomma wäre nur etwa um Faktor 2 schneller gewesen) nur rund 0,2 kHz erreicht werden, was für den vorgegebenen Regler nicht einmal zur stabilen Regelung ausreichte [7].

Literatur

[1] *Franklin, G. F., Powell, J. D.*: Digital Control. Reading, Mass., Addison-Wesley, 1980.
[2] *Isermann, R.*: Digitale Regelsysteme. Berlin, Springer 1977.
[3] *Ackermann, J.*: Abtastregelung. Berlin, Springer 1972.
[4] *Fliege, N.*: Digitale Filter mit dem Signalprozessor 2920. ELEKTRONIK 1981, H. 3, S. 81...85, und H. 4, S. 89...94.
[5] *Müller, K. H.*: Echtzeit-Simulation mit dem Analog-Prozessor 2920. ELEKTRONIK 1981, H. 7, S. 95...98.
[6] 2920 Analog Signal Processor Design Handbook. Intel Corp. 1980.
[7] *Wortmann, J.*: Einsatz eines Mikrorechner/Arithmetikprozessor-Systems für digitale Regelung/Filterung. Diplomarbeit Universität – GH – Paderborn, FB 10, Automatisierungstechnik, 1981.

Ulrich Gauda

Einfach und kostengünstig: Lösungen mit den TMS320-Peripherie-Bausteinen TMS32050/51

Mit der Einführung des 32-Bit-Signalprozessors TMS32010, der seine Leistungsfähigkeit bei der Berechnung von Filtern, Signalerzeugung, Sprachanalyse und Sprachsynthese sowie auf dem Gebiet der Regelungstechnik bewiesen hat, wurde es möglich, digitale Signalverarbeitung preisgünstig und kompakt zu verwirklichen. Der folgende Bericht beschreibt die beiden intelligenten Analog-Pheripherie-Bausteine TMS32050 und TMS32051, die die Schnittstelle zwischen Analogsignalen und digitaler Signalverarbeitung darstellen. Die ICs ersetzen A/D- und D/A-Umsetzer, die zugehörigen Filter und die Steuerungslogik. Zusätzlich sind beide Analog-Pheripherie-Bausteine mit einem FIFO-Speicher (8 × 16 Bit) ausgerüstet, die es dem Anwender erlauben, längere Zwischenrechnungen ohne Unterbrechung durchzuführen.

Der Typ TMS32050 ist ein programmierbarer analoger Eingangskanal („Analog Input Channel", AIC) zur Datenerfassung, der Typ TMS32051 ist ein programmierbarer Ausgangskanal („Analog Output Channel", AOC), der ein komplettes Ausgangssystem zur Erzeugung analoger Signale ersetzt. Besonders geeignet sind der TMS32050 und TMS32051 für Anwendungen im Bereich Telekommunikation (Modem), der digitalen Sprachverarbeitung sowie in der Steuerungs- und Regelungstechnik.

Digital-Signal-Verarbeitungssystem

Bild 1 zeigt die Blockschaltung für ein komplettes System zur digitalen Signalverarbeitung mit dem TMS32050/51 und dem Signalprozessor TMS32010. Der AIC besteht aus einem Antialiasing-Tiefpaß-Filter, einem Analog-Digital-Umsetzer (ADC) und einem First-In-First-Out-Speicher (FIFO).

Der TMS32010 steuert den TMS32050 über den 16-Bit-Datenbus und programmiert die Abtastrate sowie die Eckfrequenz des Tiefpaß-Filters. Ein Ausgangswert des ADC wird im FIFO-Speicher abgelegt und kann dort vom TMS32010 über den 16-Bit-Datenbus gelesen werden.

Der AOC besteht aus einem FIFO-Speicher, dem Digital-Analog-Umsetzer (DAC) und einem sin(x)/x-Korrekturfilter sowie einem Tiefpaß-Filter. Der TMS32010 lädt den FIFO-Speicher und programmiert wie beim AIC die Abtastrate sowie den Tiefpaß-Filter über den 16-Bit-Datenbus.

Beide Schaltkreise benötigen Versorgungsspannungen von ±5 V und sind TTL-kompatibel. Separate Anschlüsse für Analog- und Digitalspannungsversorgung zur Entkopplung sind vorhanden. Der interne zeitliche Ablauf des TMS32050 und TMS32051 muß synchron mit dem Prozessor TMS32010 arbeiten. Zur Synchronisierung dient das (5-MHz-)„CLOCK-OUT"-Signal des TMS32010.

Wichtigste Merkmale des AIC TM32050

- Zwei Analogeingänge
- Antialiasing-Filter, programmierbar, Eckfrequenz 3,3 kHz bis 8,1 kHz
- 16-Bit-A/D-Umsetzer-Zweierkomplement
- Abtastfrequenz max. 16 kHz, programmierbar
- Fehler ± 0,5 LSB
- FIFO-Speicher, 8 Worte à 16 Bit
- Serielles Ausgaberegister
- Versorgungsspannung ± 5 V
- 40poliges Gehäuse

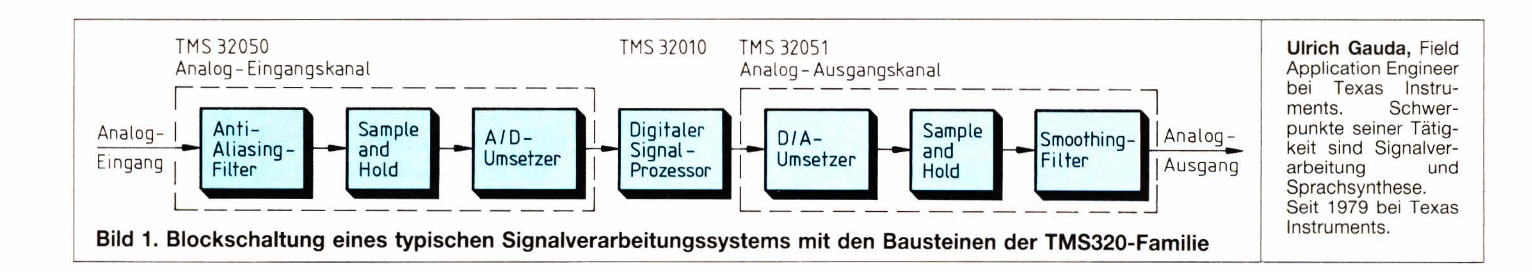


Bild 1. Blockschaltung eines typischen Signalverarbeitungssystems mit den Bausteinen der TMS320-Familie

Ulrich Gauda, Field Application Engineer bei Texas Instruments. Schwerpunkte seiner Tätigkeit sind Signalverarbeitung und Sprachsynthese. Seit 1979 bei Texas Instruments.

Architektur des AIC

Die Blockschaltung des AIC TMS32050 ist in *Bild 2* dargestellt. Beide Analog-Eingänge werden über einen Multiplexer entweder über den Antialiasing-Filter oder direkt mit dem Analog-Digital-Umsetzer (ADC) verbunden. Der interne Controller des AIC ermöglicht dem TMS32010, über die Steuerleitungen und den 16-Bit-Datenbus die Eckfrequenz des Antialiasing-Filters von 3,3 kHz bis 8,1 kHz und die Abtastfrequenz des ADC von maximal 16 kHz zu programmieren.

Der Antialiasing-Filter besteht aus drei hintereinandergeschalteten Tiefpaß-Filtern. Das Teiler-Register teilt die Takt-Frequenz (5 MHz) auf die programmierte Eckfrequenz des Filters.

Beim ADC handelt es sich um einen 16-Bit-Umsetzer, dieser liefert einen Digitalwert im Zweier-Komplement mit einem Fehler von 0,1 % und arbeitet entweder im Dauerbetrieb (programmierbar durch den TMS32010) oder im Triggermode. Im Freilaufbetrieb wird die Abtastrate durch den 10-Bit-Frequenzteiler bestimmt. Die erlaubt 1023 mögliche ADC-Perioden, wobei die maximale Abtastfrequenz bei 16 kHz liegt. Während einer Umsetzung kann der TMS32010 den Frequenzteiler um einen Wert erhöhen oder erniedrigen, um eine Phasenkorrektur durchzuführen. Im „Trigger-Mode" wird der ADC durch ein externes Triggersignal gesteuert. Das 16-Bit-Ergebnis des Analog-Digital-Umsetzers wird im 8 × 16-Bit-FIFO-Speicher abgelegt. Zwei maskierbare Statusbits FE (FIFO Empty), FF (FIFO Full) zeigen den Zustand des Zwischenspeichers an. Diese Information kann über die beiden Statusleitungen abgefragt oder auch als direktes Interrupt-Signal für den TMS32010 verwendet werden.

Architektur des AOC

Die Architektur des AOC ist dem AIC ähnlich (*Bild 3*). Dieser Baustein besteht aus einem FIFO-Speicher, einem Digital-Analog-Umsetzer (DAC), einem sin(x)/x-Korrekturfilter und einem Tiefpaß-Glättungsfilter. Wie beim AIC kann der TMS32010 die Abtastfrequenz und den Tiefpaßfilter des AOC programmieren und einen der beiden Analogausgänge selektieren. Der FIFO-Speicher und die Statusleitungen des TMS32051 sind mit denen des TMS32050 identisch. Analoge Ausgangssignale werden erzeugt, wenn der TMS32010 Daten über den 16-Bit-Bus in den 8 × 16-Bit-FIFO-Speicher schreibt.

Der DAC ist ein vorzeichenabhängiger Umsetzer mit einer Linearität von 16 Bit. Digital-Analog-Umsetzungen werden entweder im Dauerbetrieb oder im Trigger-Mode durchgeführt, wobei die Abtastrate wie beim ADC programmierbar ist. Im Dauerbetrieb wird der AOC durch das externe CVT/REF-Signal gesteuert. Zusätzlich dient das Referenz-Signal zur Phasenkorrektur. Falls dieses Signal auf High wechselt, bevor die Analog-Digital-Umsetzung begonnen hat, wird die Abtastfrequenz für einen Zyklus um einen Wert erniedrigt. Im anderen Fall,

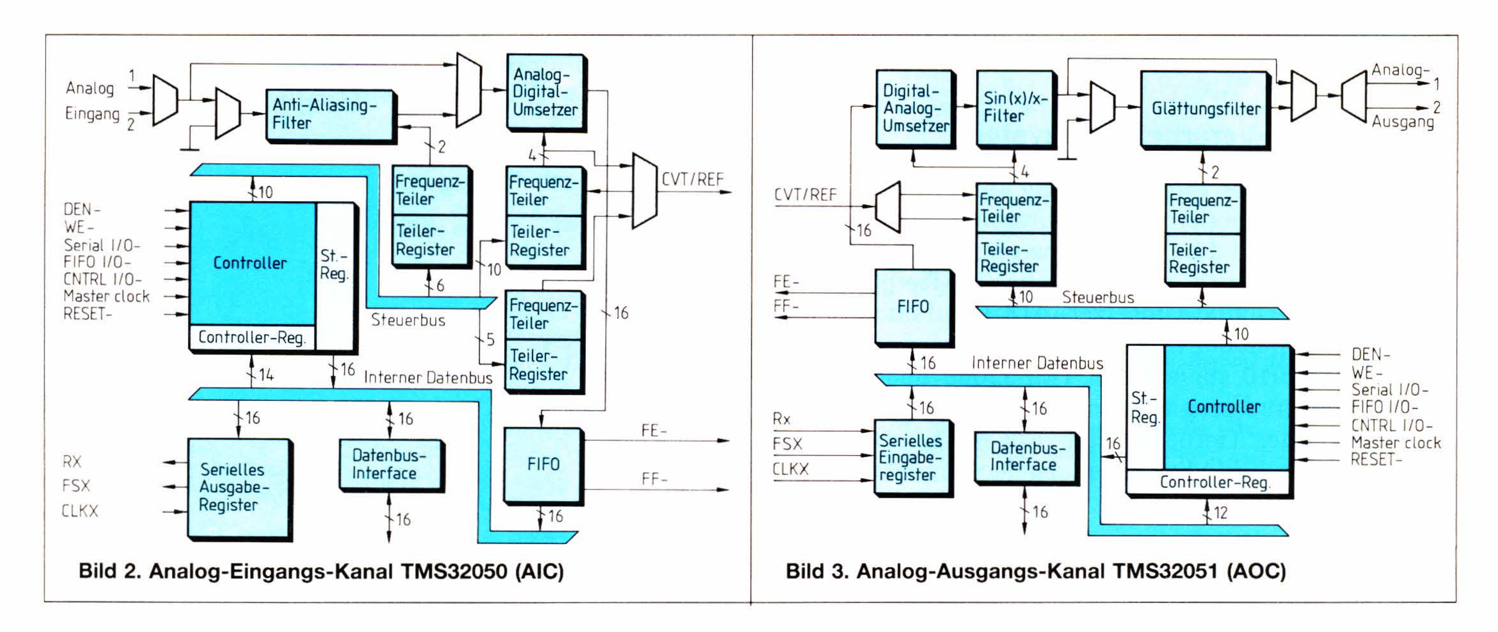


Bild 2. Analog-Eingangs-Kanal TMS32050 (AIC)

Bild 3. Analog-Ausgangs-Kanal TMS32051 (AOC)

wenn das Referenz-Signal nach einem DAC-Zykus auf High geht, wird die Abtastfrequenz für eine Wandlungsrate um eins erhöht. Zusätzlich ist es möglich, den TMS32051 im Dauerbetieb anzuhalten.

Durch Schreiben eines Befehls in das Steuerregister wird der AOC in den Sperrmode versetzt, d. h. es wird keine D/A-Umsetzung bis zur ersten abfallenden Flanke des Referenz-Signals mehr ausgeführt. Danach setzt der DAC wieder kontinuierlich um. Im Trigger-Mode wird eine D/A-Umsetzung entweder vom TMS32010 oder durch ein externes Signal gestartet, das den gleichen Eingang verwendet wie das Referenz-Signal im Dauerbetrieb. Das Glättungsfilter ist identisch mit dem Antialiasing-Filter; es läßt sich beliebig in den Ausgangskanal schalten.

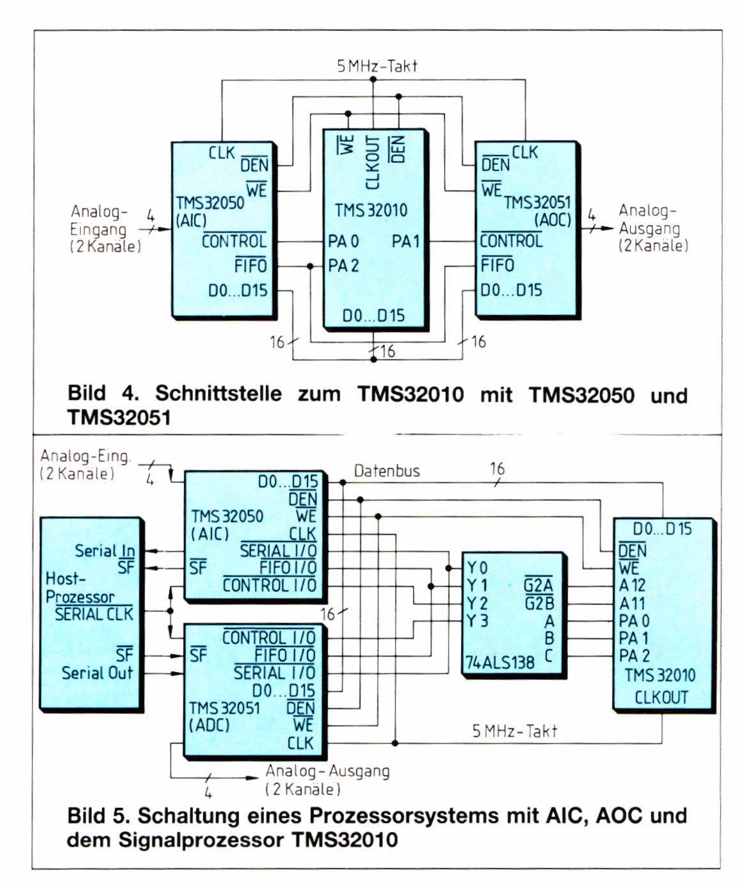


Bild 4. Schnittstelle zum TMS32010 mit TMS32050 und TMS32051

Bild 5. Schaltung eines Prozessorsystems mit AIC, AOC und dem Signalprozessor TMS32010

Serielle Schnittstelle

Die serielle Schnittstelle des TMS32050 ist als Ausgang und beim TMS32051 als Eingang ausgelegt. Diese zusätzliche Schnittstelle vereinfacht die Schaltungsauslegung zwischen einem Prozessor (z. B. Host-Prozessor) und den beiden Analog-Peripherie-Bausteinen.

Die serielle Schnittstelle erlaubt es, ein 8-Bit- oder 16-Bit-Datenwort zu senden bzw. zu empfangen. Die Wortlänge ist programmierbar. Beim 8-Bit-Datenwort werden die acht niederwertigen Datenbits angesprochen. Grundsätzlich wird als erstes das höchstwertige Bit (8. oder 16. Bit) ausgegeben. Zum Beispiel ist beim AIC bei einem 8-Bit-Wort das erste serielle Bit D7 und D0 das letzte Bit. Den Datentransfer steuern die drei Signalleitungen RX, FSX und CLKX.

Komplettes Prozessor-System

Die Analog-Interface-Bausteine TMS32050 und TMS32051 sind kompatibel mit der Busstruktur des Signalprozessors TMS32010. *Bild 4* zeigt das Interface zwischen dem TMS32050 und TMS32051. Die Leitungen DEN und WE steuern den Datentransfer von und zum Signalprozessor. Diese beiden Signale werden durch die Befehle IN und OUT des TMS32010 erzeugt. Die drei Portadreßleitungen PA0 bis PS2 selektieren den Datentransfer zwischen dem FIFO, der Seriellenschnittstelle und dem Steuerregister.

Bild 5 zeigt eine typische Anwendungsschaltung mit dem TMS32010, TMS32050 und TMS32051. Die Kommunikation zu einem Host-Prozessor geschieht über den seriellen Eingang und den seriellen Ausgang des AIC und AOC.

Realisierung vereinfacht

Die Möglichkeiten und Vorteile digitaler Signalverarbeitung sind hinreichend bekannt. Der TMS32050 und TMS32051 als erste intelligente Analog-Peripherie-Schaltkreise entsprechen den Anforderungen analog arbeitender Bausteine in vielen digital arbeitenden Systemen. Ein digitaler Aufbau ist mit dem AIC und AOC gegenüber einer äquivalenten Lösung des gleichen Problems möglich, ohne daß ein großer Aufwand an Abgleich von Bauelementen notwendig ist. Dies vereinfacht das Design, verkürzt die Entwicklungszeit und spart Kosten bei der Realisierung neuer Systeme.

Wichtigste Merkmale des AOC TMS32051

- FIFO-Speicher, 8 Worte à 16 Bit
- Serielles Eingaberegister 16-Bit-D/A-Umsetzer, Zweierkomplement
- Genauigkeit ± 0,5 LSB
- sin(x)/x-Ausgangsfilter, programmierbar
- Glättungsfilter, programmierbar
- Zwei Analogausgänge
- Versorgungsspannung ± 5 V
- 40poliges Gehäuse

Wichtigste Merkmale des Signalprozessors TMS32010

- Hardware:
  Instruktions-Zyklus 200 ns
  Wortlänge für Daten und Befehle 16 Bit
  Programm-ROM 3 KByte
  Externer Programmspeicher 8 KByte
  Datenspeicher 288 Byte
- Rechenwerk:
  Zweierkomplement-Arithmetik
  16 × 16-Bit-Multiplizierer (200 ns)
  32-Bit-Rechenwerk
  32-Bit-Akkumulator
- Ein-/Ausgabe:
  16-Bit-Datenbus. Transfer bis 40 MBit/s
  Interrupt-Leitung
  Testbare Statusleitung

Dipl.-Ing. Hans-Günter Göring

Digitale Filter in der Praxis testen

Digitale Filter finden immer breiteren Einsatz, denn ihre Vorzüge sind bestechend. Eine Hemmschwelle bilden aber immer noch die Forderung nach völligem Umdenken beim Entwurf sowie der Mangel an anschaulichen Einführungsmodellen. Das hier vorgestellte Entwurfssystem nimmt nun dem Anwender einen großen Teil der Arbeit ab und erleichtert so den Einstieg in diese neue, vielversprechende Technik.

1 Die Vorteile digitaler Filter

Die digitalen Filter – grundsätzlich immer aufgebaut wie in *Bild 1* – sind den herkömmlichen analogen in folgenden Punkten überlegen:
- keine Temperaturabhängigkeit
- keine Alterung
- keine Exemplarstreuungen
- Parameter per Software änderbar
- exakte Übereinstimmung der Theorie mit der Praxis bei Betragsfrequenzgang, Phasengang, Quantisierungsgeräusch usw.

Speziell FIR-Filter (Finite Impulse Response, endliche Impulsantwort) zeichnen sich aus durch
- einfache Entwurfsalgorithmen zur Berechnung der Filterkoeffizienten
- uneingeschränkte Stabilität.

2 Das „Filter Design Kit“

Trotz ihrer Vorteile wird die Digitaltechnik noch nicht in dem Maße eingesetzt, wie es technisch sinnvoll wäre. Der analog denkende Praktiker sieht sich oft zu großen Problemen gegenüber:
- Technik und Denkweise sind neu zu erlernen
- Einarbeitung in die Theorie der digitalen Signalverarbeitung erscheint problematisch
- Anschauliche Einführungsmodelle sind notwendig.

Die Theorie reicht allein nicht aus. Praktische Erfahrungen sind notwendig, um die neue Technik zu erlernen. Das im folgenden vorgestellte „Filter Design Kit“ (FDK) von der Firma MEDAV (*Bild 2*) verkürzt die Einarbeitungszeit entscheidend. Ohne Vorkenntnisse können wesentliche Effekte, Möglichkeiten und Grenzen der digitalen Filtertechnik studiert werden.

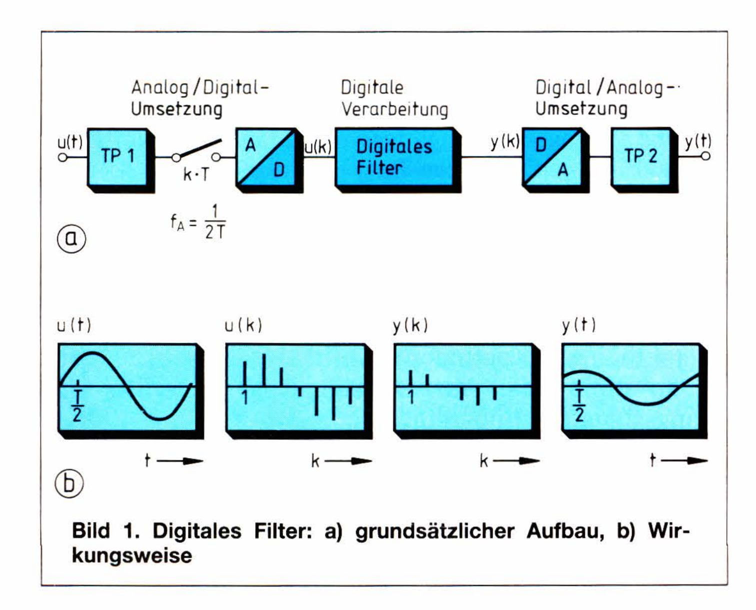


Bild 1. Digitales Filter: a) grundsätzlicher Aufbau, b) Wirkungsweise

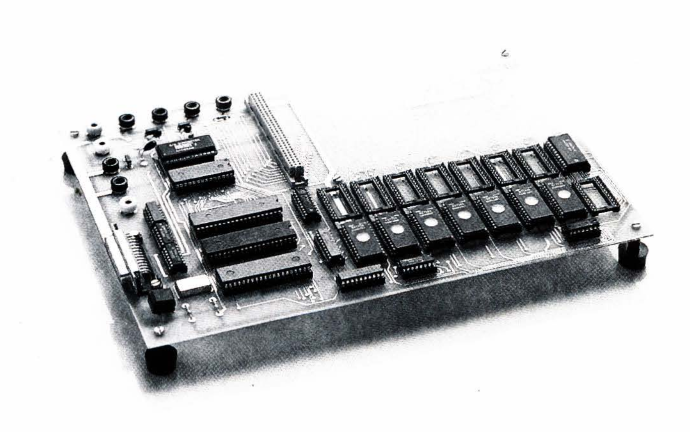


Bild 2. „Filter Design Kit“, praktische Ausführung (Foto: Medav)

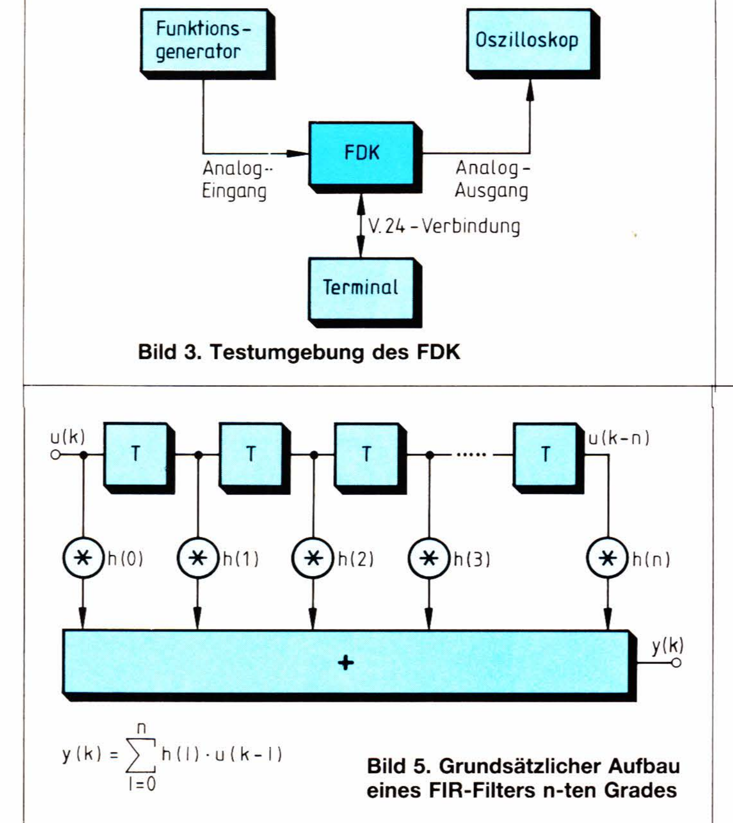


Bild 3. Testumgebung des FDK

Bild 5. Grundsätzlicher Aufbau eines FIR-Filters n-ten Grades

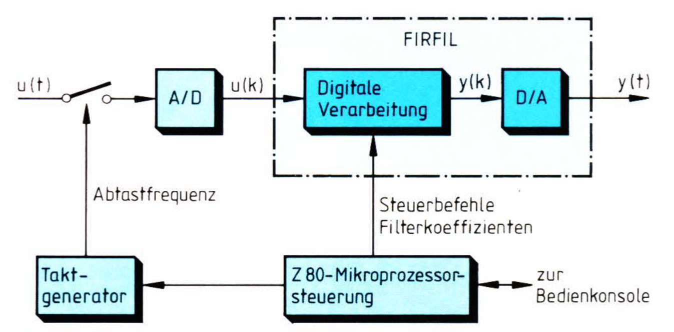


Bild 4. Blockschaltung des „Filter Design Kit". Der umrahmte Teil, hier FIR-Filter, kann ausgewechselt werden

Bild 3 zeigt eine typische analoge Testumgebung für das FDK. Die Reaktion eines digitalen Filters auf ein definiertes Testsignal wird mit dem Oszilloskop untersucht. In *Bild 4* sind beide Baugruppen des FDK dargestellt:
– FDK-Systemboard und
– Digitales Filter FIRFIL (umrahmter Teil).

Abtastfrequenz 19 200 kHz   Filter 1

Befehlsmenü FDK

0 = Demonstrationsprogramm
1 = Abtastfrequenz festlegen
2 = Filterblock einstellen
3 = Mittelungsfilter programmieren
4 = Selektives Filter programmieren
5 = eigene Koeffizienten programmieren
6 = Koeffizientensatz auf Konsole ausgeben
7 = Koeffizientensatz ändern
8 = fortlaufende Umschaltung der Filter
9 = Filterabschätzung

Eingabe:

Bild 6. Befehlsmenü des FDK nach dem Einschalten

Ein Analog/Digital-Umsetzer setzt das analoge Eingangssignal u(t) in eine Zahlenfolge u(k) konstanter Periode (Abtastfrequenz) um. Ist die Abtastfrequenz größer als die doppelte Bandbreite des Analogsignals, stellt u(k) das digitale Abbild des Analogsignals dar. Eine Recheneinheit (Digitale Verarbeitung) berechnet aus der kontinuierlichen Zahlenfolge u(k) nach einem festen Algorithmus die Ausgangszahlenfolge y(k). Ein nachfolgender Digital/Analog-Umsetzer bildet hieraus das entsprechende Analogsignal y(t). Das Systemverhalten (Tiefpaß usw.) hängt nur von der Wahl eines geeigneten Algorithmus ab.

Hier verdeutlicht sich die Flexibilität digitaler Signalverarbeitungssysteme. Die Veränderung eines Koeffizientensatzes im RAM bewirkt die Realisierung verschiedener Übertragungskennlinien, wie z.B. Hochpaß, Tiefpaß, Bandpaß, Entzerrer, Integratoren, Differenzierer usw. auf einer Hardware. Den Koeffizientensatz und die Abtastfrequenz legt die einfache Z80-Mikroprozessorsteuerung fest. Die Bedienung und Eingabe aller Befehle erfolgt im FDK menügesteuert über ein Terminal.

3 Funktionsprinzip des FIR-Filters

Bild 5 zeigt den grundsätzlichen Aufbau eines FIR-Filters. Ein Ausgangswert y(k) berechnet sich als eine Summe verzögerter Eingangswerte u(k), die jeweils mit einem Filterkoeffizienten h(n) gewichtet sind. Die Verzögerung T entspricht genau der Verzögerung um einen Abtasttakt.

Einfache Beispiele:

– *Identische Übertragungsfunktion*
Setzt man alle Koeffizienten h(n) zu Null, nur h(0) auf 1, so stimmt das Ausgangssignal y(k) mit dem Eingangssignal u(k) überein.

– *Verzögerer*
Sind alle Koeffizienten gleich Null und nur h(n) gleich 1, so erhält man ein Totzeitglied mit einer Verzögerung um n Abtastintervalle. Es gilt dann:

```
Eingabe: 4

Programmiert wird ein „ideales selektives Filter
Eingabe der Filterlänge (Integer): 100

Wahl der Fensterfunktion:

  1 = Rechteck-Fenster
  2 = Dreieck-Fenster
  3 = Hamming-Fenster
  4 = allgemeines Hamming-Fenster
  5 = Hamming-Fenster
  6 = Kaiser-Fenster
Eingabe: 1

Wahl des Filtertyps

  1 = Tiefpaß
  2 = Hochpaß
```

Bild 7. Wahl der Fensterfunktion

```
Wahl des Filtertyps

  1 = Tiefpaß
  2 = Hochpaß
  3 = Bandpaß
  4 = Bandsperre
Eingabe: 3

Eingabe untere Grenzfrequenz (Real): 0.2

Eingabe obere Grenzfrequenz (Real): 0.3

##### Koeffizienten geladen #####

Weiter mit „RETURN“
```

Bild 8. Wahl des Filtertyps und Eingabe der Grenzfrequenzen

y(t) = u(t–n·T)
mit n = Index des Koeffizienten (> 0)
T = Periode des Abtasttaktes

4 Arbeiten mit dem „Filter Design Kit“

Um auch Benutzern, denen digitale Signalverarbeitung bisher ein Fremdwort war, eine spielerisch leichte Einarbeitung zu ermöglichen, wurde beim FDK sehr viel Wert auf Bedienungskomfort gelegt. Erreicht wurde dies durch ein umfangreiches Softwarepaket (ca. 25 KByte). Nach dem Einschalten meldet sich das FDK auf dem Bildschirm des angeschlossenen Rechners mit einem Befehlsmenü (*Bild 6*). Der Anwender wählt die gewünschte Funktion aus. Hieraus einige Beispiele:

4.1 *Filterentwurf*

Die Eingabe von „4“ ruft ein Filterentwurfsprogramm für selektive Filter auf. Als nächstes erfolgt die Festlegung des Filtergrades (hier: „100“, *Bild 7*). Die Filterkoeffizienten können mit verschiedenen Fensterfunktionen gewichtet werden. Sie beeinflussen die Feinheiten des Frequenzgangs (Welligkeit, Steilheit). Mit „3“ wählt man den Filtertyp „Bandpaß“ aus (*Bild 8*). Zum Abschluß gibt man noch die Grenzfrequenzen des Durchlaßbereiches ein, angegeben als Bruchteil der Abtastfrequenz.

Den Frequenzverlauf für diesen Bandpaß mit verschiedenen Fensterfunktionen zeigen die *Bilder 9* und *10*.

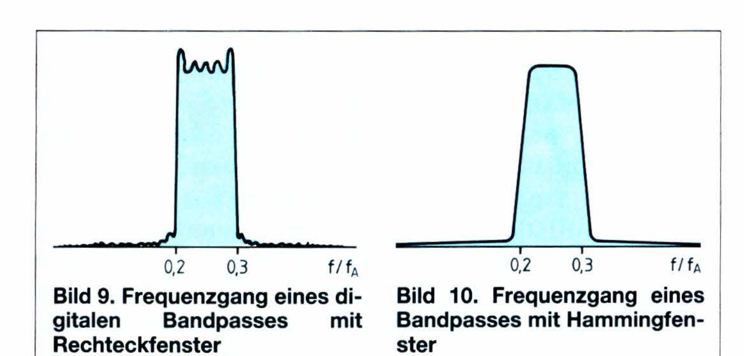
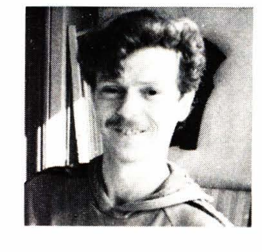


Bild 9. Frequenzgang eines digitalen Bandpasses mit Rechteckfenster

Bild 10. Frequenzgang eines Bandpasses mit Hammingfenster



Dipl.-Ing. Hans Günter Göring, geboren in Vaihingen/Enz, aufgewachsen in Nürnberg (Franken), studierte an der Friedrich-Alexander-Universität in Erlangen Elektrotechnik. Sein Studienschwerpunkt war Digitale Signalverarbeitung. Seit 1983 beschäftigt er sich bei der MEDAV GmbH in Buckenhof mit Problemen der Signalverarbeitung und ihrer Lösung durch spezielle Hard- und Software.

```
    6 = Koeffizientensatz auf Konsole ausgeben
    7 = Koeffizientensatz ändern
    8 = fortlaufende Umschaltung der Filter
    9 = Filterabschätzung

  Eingabe: 9

Berechnung des Filtergrades FIR-TP-Filter

Eingabe Abtastfrequenz in kHz               (Real): 10.0
Eingabe Durchlaßgrenzfrequenz in kHz        (Real):  2.4
Eingabe Sperrgrenzfrequenz in kHz           (Real):  2.6
Eingabe max. Dämpfung Durchl. in dB         (Real):  0.1
Eingabe min. Dämpfung Sperrb. in dB         (Real): 60.

Resultierende Steilheit in dB pro Oktave         : 518.7155
(entsprechendes Analogfilter)
Minimaler Filtergrad (FOURIER, KAISER-Fenster)   : 181
Minimaler Filtergrad (TSCHEBYSCHEFF)             : 135

Weiter mit „RETURN“
```

Bild 11. Filterabschätzung mit dem FDK

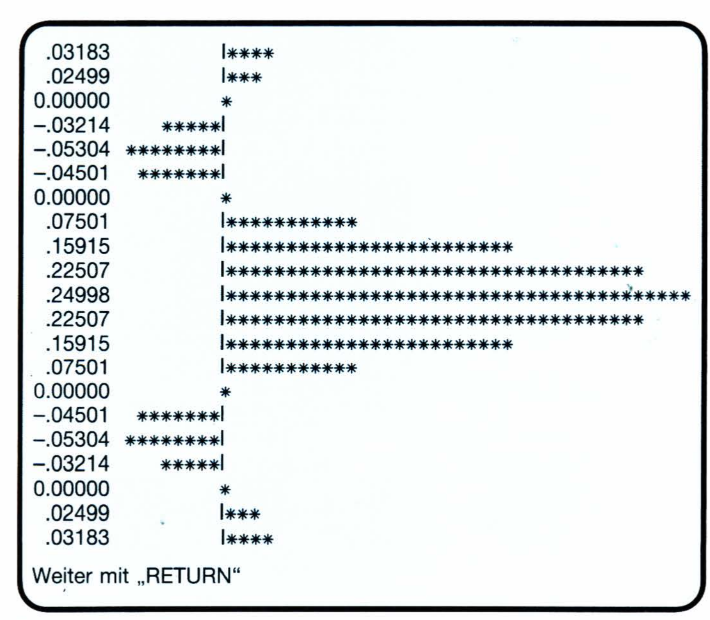


Bild 12. Tiefpaß-Impulsantwort

4.2 Filtergradabschätzung

Das Ergebnis einer Filtergradabschätzung (Programm „9“) stellt *Bild 11* dar. Die Leistungsfähigkeit des Digitalsystems erläutert das gewählte Frequenzgangbeispiel mit einer Steilheit im Übergangsbereich von ca. 500 dB/Oktave bei linearem Phasengang.

4.3 Darstellung von Impulsantworten

Umfangreiche Tabellenwerke zur Beschreibung selektiver Filter (TP, HP, BP, BS) erübrigen sich durch implementierte Entwurfsalgorithmen. Neben dem unmittelbaren Programmieren des Filterbausteines „FIRFIL“ ist die Ausgabe berechneter Koeffizientensätze möglich. *Bild 12* zeigt die Impulsantwort eines Tiefpaß-Filters mit 21 Koeffizienten.

Literatur

[1] *Kolb, H. J.:* Digitales FIR-Filter für die Meßwertverarbeitung. ELEKTRONIK 1983, H. 3, S. 85...88.

[2] *Tiefenthaler, C.:* Digitale Transversalfilter. ELEKTRONIK 1984, H. 1, S. 57...64.

[3] *Stearns, S. D.:* Digitale Verarbeitung analoger Signale. Oldenbourg Verlag, München, Wien, 1979.

[4] *Fliege, N.:* Digitale Filter mit dem Signalprozessor 2920. ELEKTRONIK 1981, H. 3, S. 81...85 und H. 4, S. 89...94.

[5] *Dischinger, T., Nielinger, H.:* Linearphasige FIR-Filter mit Signalprozessor realisiert. ELEKTRONIK 1983, H. 26, S. 53...56.

Dr. rer. nat. Thomas Fischer

Digitale Signalverarbeitung in der Konsumelektronik

Digitale Signalverarbeitung in Echtzeit ist ein ganz großes Schlagwort geworden. Was verbirgt sich dahinter? Was ist daran so neu? Was unterscheidet die digitale Signalverarbeitung von der herkömmlichen EDV bzw. der Mikroprozessortechnik? Auf alle diese Fragen gibt der folgende Beitrag eine Antwort. Er zeigt darüber hinaus auf, wie diese Technik für die Konsumelektronik nutzbar gemacht werden kann.

1 Einführung

Die Vorteile der Digitaltechnik brauchen hier nicht erneut diskutiert zu werden. Es ist hinreichend bekannt, was die Digitaltechnik für Langzeitstabilität, Integrierbarkeit, Toleranz gegenüber Parameterstreuungen, Speicherbarkeit der Signale und Rauschverhalten bedeutet. Diese Vorteile werden gerade jetzt in einer ungemein dynamischen Entwicklung für die Konsumelektronik erschlossen. Drei Einsatzgebiete seien zur Verdeutlichung erwähnt: synthetische Spracherzeugung, digital arbeitende Fernsehgeräte und digitale Schallplatten.

Was unterscheidet die herkömmliche analoge und die digitale Signalverarbeitung? Der entscheidende systemtheoretische Unterschied ist der zwischen einem abgetasteten und einem kontinuierlichen System (*Tabelle 1*). Die Abtastrate begrenzt die Verarbeitungsbandbreite des Systems. In einem abgetasteten System können nur Signale verarbeitet werden, deren höchste Frequenzkomponente die halbe Abtastfrequenz nicht überschreitet (Nyquist-Theorem). Höhere Frequenzanteile erscheinen als falsche Frequenzen (unter einem „Alias“) im Basisband. Diese Störungen werden Aliasing genannt.

In digitalen Systemen lassen sich immer wiederkehrende Funktionseinheiten identifizieren (*Bild 1*). Dies sind Anti-Aliasing-Filter, Analog-Digital-Umsetzer, Prozessor, Digital-Analog-Umsetzer und Rekonstruktionsfilter. Letzteres interpoliert die Werte zwischen den Taktzeiten, denn nur zu den Taktzeiten steht der richtige Wert am D/A-Umsetzer an. Gleichzeitig werden die hochfrequenten Störanteile ausgefiltert, die durch die scharfen Sprünge des Umsetzers entstehen. Die Anordnung des Systems ist nicht immer genauso. Es gibt Fälle, in denen mit sehr hoher Rate abgetastet wird, der Signalprozessor aber mit sehr viel kleinerer Systemfrequenz arbeitet. In einem solchen Fall kann das Anti-Aliasing-Filter zwischen Umsetzer und Prozessor angeordnet und als digitales Filter ausgebildet sein. Umgekehrt ist das Rekonstruktionsfilter oft in nachgeordneten Systemkomponenten untergebracht, z. B. wird die Tiefpaßwirkung von Lautsprechern oder die begrenzte Bandbreite der Leistungsverstärker ausgenutzt.

Die digitale Verarbeitung von Signalen begann mit der Einführung moderner digitaler Rechenmaschinen. Das war zunächst nicht in Echtzeit möglich. Eine der ersten Anwendungen war die Analyse von (aufgezeichneten) seismischen Signalen. Eine spätere berühmte Anwendung war die Filterung und Bearbeitung der Bilder, die von Satelliten und Raumsonden auf die Erde zurückgesendet wurden. Diese Art der digitalen Signalverarbeitung geschah und geschieht auf digitalen Datenverarbeitungsanlagen. Mit der Einführung schneller LSI-Bausteine war es möglich, digitale Signale in Echtzeit zu verarbeiten. Zuerst waren das festverdrahtete (Hardware-)Lösungen. Heute ist man an der Schwelle der Masseneinführung programmierbarer Signalprozessoren angelangt. Sie entwickelten sich aus der Mikroprozessortechnik, und sie werden eine ähnliche Revolution in der Signalverarbeitung bewirken wie die Mikroprozessoren in der Datenverarbeitung.

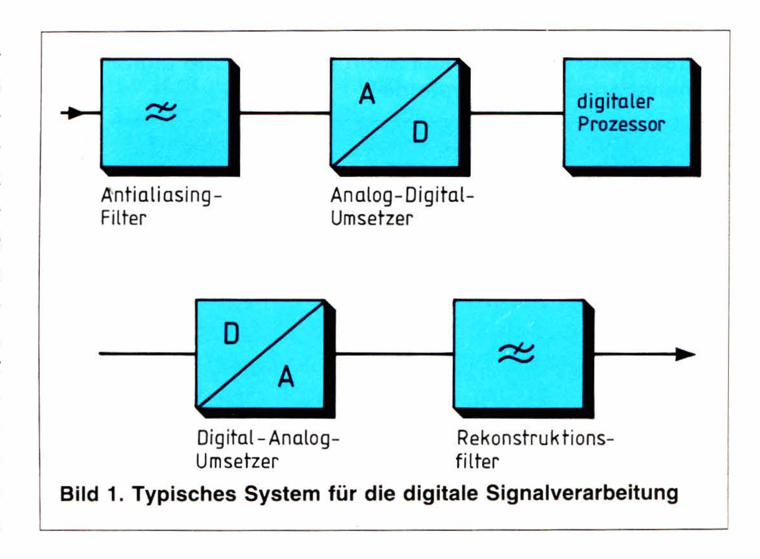


Bild 1. Typisches System für die digitale Signalverarbeitung

2 Signalprozessor gegenüber Mikroprozessor

Was unterscheidet einen Signalprozessor von einem Mikroprozessor? Warum ist die digitale Signalverarbeitung eine neue Disziplin, und warum führt man digitale Signalprozessoren unter eigenem Namen als neue Klasse von Maschinen? Informationsverarbeitende Maschinen (EDV-Anlagen, Mikrocomputer) und signalverarbeitende Maschinen bedienen sich ähnlicher Techniken; beide beruhen auf digitalen Schaltungen. In *Tabelle 2* ist der Versuch unternommen, die wichtigsten Charakteristiken für beide Klassen zusammenzustellen. Der wichtigste Unterschied ist, daß informationsverarbeitende Systeme Informationen, also Daten, Texte, Zahlen bearbeiten, während signalverarbeitende Systeme den Träger von Informationen, nämlich (elektrische) Signale bearbeiten, eventuell die Information aus dem Signal gewinnen und an ein informationsverarbeitendes System weiterreichen.

Die Entscheidung über den Einsatz digitaler Signalverarbeitung wird durch verschiedene Überlegungen beeinflußt. Der größeren Komplexität digitaler Schaltungen steht die größere Dichte von Elementen auf dem IC und die Einsparung teurer und anfälliger analoger Komponenten gegenüber. Die leichte Speicherbarkeit digitaler Daten macht manche Anwendungen überhaupt erst möglich. Erzielbarer Preis/Stückzahl/Bandbreite/Auflösung beeinflussen die Entscheidungen ja/nein, festverdrahtet/programmierbar oder Standardbausteine/spezielle VLSI-Schaltungen.

Dies soll an einem Beispiel erläutert werden. Bei ITT Intermetall wurde ein Schaltungskonzept entwickelt, das die gesamte Basisband-Signalverarbeitung eines Fernsehempfängers mit digitalen Schaltungen realisiert. Der IC-Satz besteht aus fünf VLSI-HMOS-Schaltungen, nämlich Video-Prozessor, Audio-Wandler, Audio-Prozessor, Ablenk-Prozessor und Steuereinheit [1].

Hinzu kommen weitere ICs in bipolarer Technik, die Aufgaben übernehmen, für die die HMOS-Technik weniger geeignet ist. Dies sind Video-Umsetzer, Taktgenerator, Infrarot-Vorverstärker und das Tuner-Interface.

3 Beispiel: Digital arbeitendes Fernsehgerät

Die Basisbandverarbeitung beginnt bei Video- und Audiodemodulator. Das Videosignal wird im Video-Umsetzer in eine Folge von Digitalzahlen mit einer Auflösung von sieben Bit umgesetzt. Die Referenzspannung des Analog-Digital-Umsetzers wird von Zeile zu Zeile um den halben Wert des kleinstwertigen Bits verändert. Bei der Betrachtung ergibt sich der Eindruck eines mit acht Bit umgesetzten und verarbeiteten Signals. Der Umsetzer selbst ist als „Flash Converter" mit 127 parallelen Komparatoren ausgebildet. Das Ausgabewort ist Gray-codiert, um Störimpulse infolge unterschiedlicher Geschwindigkeiten der Komparatoren oder des Codierers selbst auszuschalten.

Die Bandbreite des Videosignals ist ca. 5 MHz. Dies erfordert einen Abtast- und Verarbeitungstakt von mindestens 10 MHz. Wegen der Art, wie in PAL- und NTSC-Signalen die Farbe codiert ist, ist es für die Empfangsschaltung vorteilhaft, die vierfache Farbhilfsträgerfrequenz zu benutzen und den Abtasttakt phasenstarr mit dem Farbhilfsträger zu verkoppeln. Die Abtastung des Videosignals bei 45°, 135°, 225° und 315° Phasenlage führt zu einer inhärenten Trennung von (R–Y), (B–Y), –(R–Y) und –(B–Y). Die Abtastrate ist daher 17,7 MHz in einem PAL-Empfänger und 14,3 MHz in einem NTSC-Empfänger. Gleichzeitig wird der Abtasttakt als Systemtakt für alle Signalprozessoren verwendet.

Das digitalisierte Videosignal wird dem Video- und dem Ablenk-Prozessor zugeführt. Die innere Struktur dieser Prozessoren spiegelt den zugrundeliegende Datenfluß sehr genau wider. Die Ankunft eines neuen Datenworts mit jedem Takt, d. h. alle 56 ns, erfordert eine extrem sequentiell ausgelegte Architektur (Pipeline-Architektur) des Prozessors. Die einzelnen Verarbeitungsblöcke lassen sich auf dem IC deutlich unterscheiden (*Bild 2*). Aber nicht nur die Makrostruktur, auch die Mikrostruktur ist als Pipeline ausgelegt. Die Multiplizierer, die für die digitalen Filter benötigt werden, sind als Folge von Schiebe- und Addierwerken ausgelegt. Die Kunst beim Entwurf dieser Filter besteht

Tabelle 1. Gegenüberstellung charakteristischer Eigenschaften analoger und digitaler signalverarbeitender Systeme

	Analoge Signalverarbeitung	Digitale Signalverarbeitung
Verarbeitungszeit	Echtzeit	nicht in Echtzeit oder Echtzeit
Zeitverlauf	kontinuierlich	diskret (abgetastete Systeme)
System-philosophie	Hardware-orientiert	Programm-orientiert
Mathematische Hilfsmittel	Laplace-Transformation, Differentialgleichungen	z-Transformation, Differenzengleichung
Abhängigkeit von Bauelementeparametern	groß	gering

Tabelle 2. Typische Eigenschaften informationsverarbeitender und signalverarbeitender Systeme; in der Praxis sind die Übergänge oft fließend

	Informationsverarbeitende Systeme (EDV)	Signalprozessoren
Gegenstand der Bearbeitung	Daten, Texte	Signale, die Information transportieren
Typische Instruktionen	mov, +, –, vergleiche, and, or, call, bedingte Sprünge	+, –, x, mov
Art der Programme	kontrollintensiv	rechenintensiv
Programmablauf	variabel	linear, repetitiv
Darstellung der Symbole	ASCII, EBCDIC, Festkomma, Gleitkomma, BCD	Festkomma

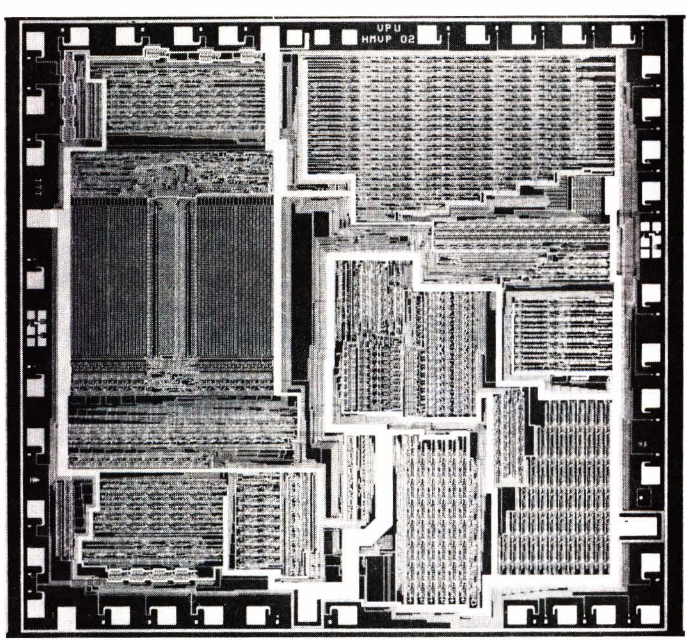


Bild 2. Chip-Foto des Video-Prozessors MAA 2200

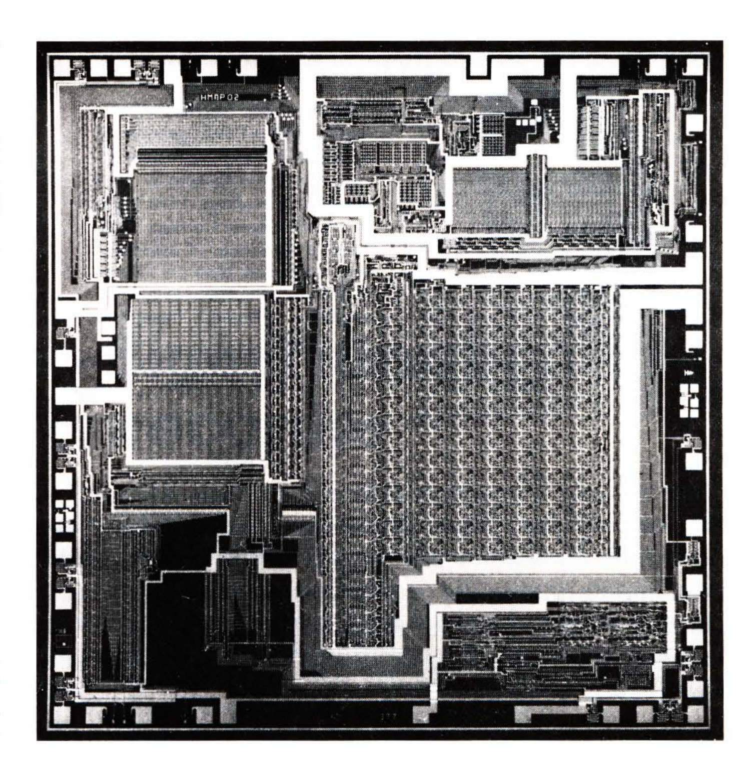


Bild 3. ▶ Chip-Foto des programmierbaren Echtzeit-Signalprozessors MAA 2400

darin, Filterstrukturen und Koeffizienten zu finden, bei denen die Anzahl der Bits mit Wert 1 minimal ist. Nur Bits mit Wert 1 erfordern eine physikalisch existierende Schiebe- und Addierstufe.

Die Phasenregelschaltung, die den Referenzträgeroszillator auf richtige Frequenz und Phase regelt, ist eine nahezu reine Digitalschaltung. Während der Übertragung der Phasenreferenz des Senders (Burst) werden die (R–Y)- und (B–Y)-Komponenten miteinander verglichen. Ihre numerische Differenz ist ein direktes Maß für die relative Phase von Abtasttakt und Burst. Sie wird benutzt, um den Referenzträgeroszillator nachzuregeln. Der Referenzträgeroszillator wird zugleich als Generator des Systemtaktes eingesetzt. Als spannungsgesteuerter Oszillator ist er das letzte analoge Glied in einer sonst rein digitalen Phasenregelschaltung.

Die Verzögerungsleitung, die für die PAL-Kompensation benutzt wird, wird bei der Verarbeitung von Farbsignalen nach NTSC-Norm zur Realisierung eines Kammfilters eingesetzt.

Der zweite Signalprozessor, der das digitalisierte Videosignal zugeführt bekommt, ist der Ablenk-Prozessor. Digitale Filter trennen Zeilen- und Bildsynchronimpulse ab. Da der Ablenk-Prozessor die Lage der Schwarzschulter erkennt, gibt er ein Steuersignal an die Klemmschaltung des Video-A/D-Umsetzers. So wird erreicht, daß immer der volle Bereich des A/D-Umsetzers ausgenutzt wird. Die hervorstechendste Eigenschaft des Ablenk-Prozessors ist die Benutzung von Abzählalgorithmen zur Erzeugung der Horizontal- und Vertikalfrequenzen. Auf diese Weise wird die Zeilen- und Bildlage mit der Präzision des Bursts stabilisiert, so daß Störimpulse durch Rauschen oder Störimpulse durch Haushaltsgeräte die Stabilität des Bildes nicht mehr beeinflussen können. Die zeitliche Auflösung des Taktes ist 56 ns. Das reicht zur Steuerung der Zeilenablenkung nicht aus. Die Genauigkeit des abgegebenen Steuerimpulses wird durch eine einstellbare Kette von Verzögerungsgliedern erhöht. Die genaue Messung von Phasenlage und Frequenz wird durch Mittelung über viele Zeilen erreicht.

Das Vertikalsignal ist pulsweitenmoduliert. Es enthält bereits die notwendigen Linearitätskorrekturen. Ein zweiter pulsweitenmodulierter Ausgang stellt die Ost-West-Parabel zur Verfügung.

Im Audio-Prozessor ist die Struktur der Signalverarbeitung verschieden von der Struktur des ICs. Das Kernstück des Audio-Prozessors ist ein schneller 16×8-Multiplizierer. Die Abtastrate für das Tonsignal ist 35 kHz, aber jeder Wert muß durch eine Vielzahl von Filtern laufen mit einer Vielzahl von variierbaren Koeffizienten. *Bild 3* zeigt ein Chip-Foto des Audio-Prozessors. Er ist ein ROM-gesteuerter Prozessor, der mit einem Ziel ausgelegt wurde, nämlich den Durchsatz durch den Multiplizierer zu maximieren. Der Instruktionszyklus ist 56 ns. Der Multiplizierer benötigt ca. 200 ns, so daß pro Multiplikation drei Befehlszyklen für Verwaltungsaufgaben zur Verfügung stehen. Der Audio-Prozessor erhält sein Eingangssignal vom Audiowandler, der nach dem Sigma-Delta-Prinzip arbeitet (Pulsdichtemodulation mit nachgeschaltetem Konversionsfilter). Die beiden Ausgangssignale – für Stereobetrieb – sind pulsweitenmoduliert. Durch Änderungen des ROMs kann der Audio-Prozessor an verschiedene Anforderungen angepaßt werden.



Dipl.-Phys. Dr. Thomas Fischer wurde 1948 in Essen geboren. Er studierte Physik; nach dem Diplom 1974 promovierte er 1978 an der Uni Freiburg. 1979 trat er in die Abteilung Concept Engineering der Firma ITT Intermetall ein, wo er seit 1980 die Gruppe Software-Entwicklung leitet. Hobbys: Skilaufen, Fotografieren, Tanzen

Die zentrale Steuereinheit koordiniert die Aktivitäten der verschiedenen Signalprozessoren. Sie empfängt die Befehle des Benutzers über Infrarot oder Direkteingabetasten und stellt die entsprechenden Register der Signalprozessoren auf die benötigten Werte ein. Dies ist Datenverarbeitung, und erwartungsgemäß ist die Steuerschaltung ein Mikrocomputer, der um einige Peripherieschaltungen erweitert wurde. Dies sind Abstimm-PLL, serielles Businterface, Infrarot-Decoder und ein nichtflüchtiger Speicher zur dauerhaften Ablage von Abstimm- und Abgleichinformationen. Die Programmierbarkeit des Mikrocomputers erlaubt es verschiedenen Herstellern, ihre Geräte nach eigenen Vorstellungen mit speziellen Eigenschaften auszustatten.

Es zeigt sich, daß für extrem schnelle (videofrequente) Signale nur festverdrahtete Schaltungen in Frage kommen. In diesem Fall erweisen sich die verhältnismäßig kurzen Worte und festen Filterkoeffizienten als Vorteil. Im Gegensatz dazu liegen beim Audio-Prozessor lange Worte und variable Filterkoeffizienten vor. Nur durch geeignete Prozessorstrukturen lassen sich zu Konsumerpreisen herstellbare ICs realisieren.

4 Weitere Anwendungen

Aber nicht nur in der Unterhaltungselektronik ist die digitale Signalverarbeitung auf dem Vormarsch. Sie wird z. B. in Automobilanwendungen und Telefonnetzen Einzug halten. Projekte, an denen gearbeitet wird, betreffen Autoradio, Zündung, automatische Getriebe und Anti-Blockier-Systeme in der Automobilanwendung oder den Einsatz für digitale Vermittlungstechnik, digitale Teilnehmerschaltung, Tonwahlverfahren, Digon (digitales Ortsnetz) und Bigfon (glasfasergestütztes Breitband-Kommunikationssystem). In diesen Anwendungen wird uns die digitale Echtzeitverarbeitung von Signalen auf Schritt und Tritt begegnen und unsere Umgebung beeinflussen.

Manche Anwendungen werden die (Arbeits-)Umgebung revolutionieren, z. B. der breite Einsatz synthetischer Sprache und Spracherkennung, die beide ohne digitale Echtzeit-Signalverarbeitung nicht vorstellbar sind. Für den Ingenieur liegt die Herausforderung in den neuen Werkzeugen technischer und theoretischer Art (siehe Tabelle 1), für den Hersteller in der Kombination neuer Techniken und Möglichkeiten (z. B. Bigfon: Optik/VLSI/Digitaltechnik). Hier „an vorderster Front mitzumischen“ wird entscheidend für die Position der deutschen Industrie im internationalen Wettbewerb sein.

Literatur

[1] *Fischer, T.*: Fernsehen wird digital. ELEKTRONIK 1981, H. 16, S. 27...35.

[2] *Fischer, T.*: Zeitdiskrete Signalverarbeitung und z-Transformation. ELEKTRONIK 1981, H. 22, S. 65...68.

[3] *Schirm IV, L.*: Digitale Signalverarbeitung mit Fließkommaarithmetik. ELEKTRONIK 1982, H. 11, S. 55...61.

[4] *Demmer, Draheim, Warmuth, Lenth*: Pipelining-Verfahren in der digitalen Signalverarbeitung. ELEKTRONIK 1982, H. 3, S. 73...78.

[5] *Klasche, G.*: Bauelemente: Noch keine Grenzen in Sicht. ELEKTRONIK 1982, H. 13, S. 75...81.

Dr.-Ing. Peter Draheim

Digitalisierung der Video-Signalverarbeitung Beispiel: Video-Kassettenrecorder

Die digitale Echtzeit-Signalverarbeitung ist seit einigen Jahren ein sehr aktuelles Thema. Insbesondere seitdem die Gatterlaufzeiten von MOS-Schaltungen so schnell geworden sind, daß sich ICs mit Taktraten von etwa 20 MHz realisieren lassen. Dadurch können Signalbandbreiten von über 6 MHz verarbeitet werden. Das öffnete dieser Technik, die bisher nur im professionellen Bereich angewandt wurde, die Tür für Konsum-Applikationen. Bisher wurde jedoch fast ausschließlich über Anwendungen in Fernsehempfängern berichtet [1]. Der folgende Beitrag beschreibt die Anwendung dieser Technik in einem Video-Kassettenrecorder des Systems V 2000, wobei zunächst die prinzipielle Verarbeitungsstruktur erläutert wird.

1 Systemeinführung

Die Signalverarbeitung in Video-Kassettenrecordern ist, unabhängig von deren Aufzeichnungsnorm VCR, VHS oder Betamax, dadurch gekennzeichnet, daß die Luminanzinformation von dem ursprünglich amplituden-modulierten FBAS-Signal zur Aufzeichnung auf das Band in ein frequenzmoduliertes Signal umgewandelt wird. Dies erfolgt wegen der höheren Störunempfindlichkeit von FM-Signalen.

Die im FBAS-Signal bei 4,43 MHz liegende Chrominanzinformation wird zur Aufzeichnung mit einem Träger so heruntergemischt, daß sie frequenzmäßig unterhalb der nun frequenzmodulierten Luminanzinformation liegt. Sie bleibt jedoch amplituden-moduliert. In *Bild 1* ist das Spektrum der aufgezeichneten Videosignale für ein VCR-Gerät des Typs V 2000 dargestellt, wobei in der Abbildung auch noch die DTF-Signale (*Dynamic Track Following*) eingezeichnet sind, die zur Band-Kopf-Steuerung benötigt werden und die frequenzmäßig noch unterhalb der Chrominanzinformation aufgezeichnet werden.

Die einzelnen Normen unterscheiden sich nun im wesentlichen in den Frequenzlagen und den Modulationshüben. Im weiteren wird hier nur auf das VCR-System V 2000 eingegangen. *Bild 2* zeigt eine sehr vereinfachte Prinzipschaltung der VCR-Aufnahme und -Wiedergabe. Für die Aufnahme wird das 5 MHz breite FBAS-Signal über ein Tiefpaß-Filter in ein 3,0 MHz breites Luminanzsignal und über einen Bandpaß in das 4,43 MHz geträgerte Chrominanzsignal aufgespalten. Das Luminanzsignal wird dann so frequenzmoduliert, daß die Synchronimpulse der unteren Hubfrequenz des Modulators von 3,3 MHz zugeordnet werden. Die Schwarzschulter liegt dann bei 3,8 MHz, die obere Hubfrequenz von 4,8 MHz stellt das Spitzenweiß dar.

Das Chrominanzsignal wird mit einem 5,06-MHz-Träger, der aus dem Burst und der Zeilenfrequenz gewonnen worden ist, auf 625 kHz heruntergemischt und dann zusammen mit der Luminanz auf das Band gegeben. Der entsprechende inverse Vorgang läuft dann wie in Bild 2 vereinfacht dargestellt für die Wiedergabe vom Band ab. Das hier sehr grob beschriebene Prinzip wird als „Colour-Under"-Verfahren bezeichnet.

2 Warum Digitalisierung?

Nach dieser kurzen Systemeinführung soll nun die digitale Realisierung des „Colour-Under"-Verfahrens für

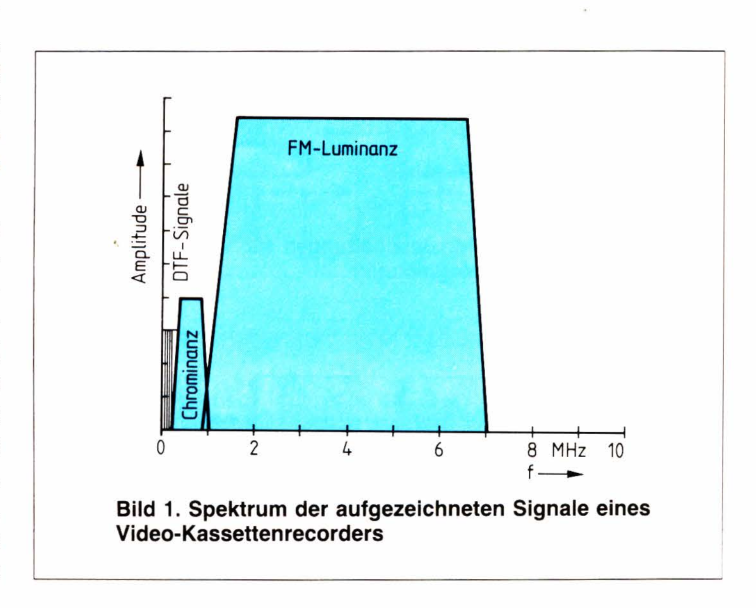


Bild 1. Spektrum der aufgezeichneten Signale eines Video-Kassettenrecorders

das V 2000-System betrachtet werden. Dabei wird für die Digitalisierung von der Randbedingung ausgegangen, daß in der bisherigen analogen Bandaufzeichnungsart und in der V 2000-Norm keinerlei Modifikationen erfolgen sollen. Das bedeutet, daß nur die Signalverarbeitung, kompatibel zu dem schon bestehenden System, digitalisiert wird.

Ausgehend von dem hier vorliegenden, sehr komplexen Signalverarbeitungssystem, so stellt sich zuerst die Frage nach dem Vorteil einer Digitalisierung. Da in der VCR-Signalverarbeitung eine mehrfache Signalumsetzung vorgenommen wird, ist dies mit einem hohen Aufwand für Filter sowie auch einigem Aufwand zur Trägererzeugung verbunden. Das heißt, es werden zur Realisierung der Signalverarbeitung neben einigen ICs eine Vielzahl von peripheren Komponenten für die Filterfunktionen benötigt, wobei die Kosten für die Peripherie und deren Abgleich die IC-Kosten bei weitem überschreiten. Diese Situation wird auch bei einer weiteren Zusammenfassung der IC-Funktionen nicht wesentlich verbessert, da die Filterfunktionen in analoger Schaltungstechnik zur Zeit nicht monolithisch mit den bipolaren ICs integriert werden können. Dies kann nur durch den Übergang in eine digitale Schaltungstechnik erreicht werden, wobei solche Arbeiten für den Fernsehbereich ebenfalls schon durchgeführt werden.

Das erste Ziel einer Digitalisierung der VCR-Videosignalverarbeitung ist somit eine wesentliche Verbesserung der Ökonomie des Systems durch Reduktion des Peripherieaufwandes.

Bild 2. Vereinfachte Prinzipschaltungen für VCR-Aufnahme und -Wiedergabe (Analogtechnik)

Bild 3. Prinzipschaltung eines digitalen FM-Modulators

3 Realisierung der digitalen Signalverarbeitung

Betrachtet man die Blockschaltung in Bild 2, so sind die zentralen Funktionselemente für eine digitale Signalverarbeitung in einem Video-Kassettenrecorder die *digitale Frequenz-Modulation*, die *Mischung* und die *Frequenz-Demodulation*. Das sind auch die drei Funktionseinheiten, in denen sich die Videosignalverarbeitung im VCR-Gerät von der des Fernsehempfängers (mit Ausnahme der Demodulation von SECAM-Signalen) unterscheidet.

3.1 *Digitaler FM-Modulator*

Der von seiner digitalen Struktur einfachste Baustein ist der FM-Modulator. *Bild 3* zeigt die Blockschaltung eines digitalen FM-Modulators. Dieser besteht aus zwei Addierern, einem Speicher und einer Sinustafel. Der erste Addierer bildet die Summe aus den digitalen Eingangsamplitudenwerten und einem konstanten digitalen Wert A, der proportional ist der unteren stationären Modulationsfrequenz.

Der zweite Addierer wirkt zusammen mit dem Speicher als Integrierer. Durch die endliche Wortbreite des Addierers kommt es bei der ständigen Addition der verzögerten Ausgangswerte zu den Eingangswerten (Integration) zu einem Überlauf. Damit entspricht das Additionsergebnis hinter dem Speicher Abtastwerten einer sägezahnförmigen Schwingung konstanter Amplitude. Die Frequenz dieser Schwingung wird durch die Eingangswerte in den Modulator streng linear beeinflußt. Ein nachfolgendes ROM, in dem eine Sinustafel abgespeichert ist, setzt die Abtastwerte der sägezahnförmigen Schwingung in die Abtastwerte einer harmonischen Schwingung um. Dieser FM-Modulator ist jedoch auch das Basisbauelement für die Erzeugung des Trägers eines digitalen Frequenzmischers, wobei die eigentliche Mischung nur mit Hilfe eines Multiplizierers geschieht. Die Trägergeneration erfolgt über einen FM-Modulator mit festem Eingangswert. Hierbei bestimmt die Wortbreite der internen Verarbeitung im FM-Modulator die Genauigkeit der Frequenz.

3.2 *Digitaler FM-Demodulator*

Der digitale FM-Demodulator ist von den drei Komponenten die aufwendigste und kritischste Schaltung. Im Gegensatz zur analogen FM-Demodulation, bei der man

zur Störunterdrückung von überlagerten AM-Signalen einen Begrenzerverstärker vorschalten kann, ist das bei einem digitalen FM-Demodulator nicht möglich. In einem Digitalwort liegt die Information in den zeitdiskretisierten und quantisierten Amplitudenwerten. Eine „Begrenzung“ würde einer Reduktion der Quantisierungsstufen bis hin zu einem Bit und damit der Auflösung entsprechen, also einem Verlust an Information.

Daneben bedeutet eine Begrenzung jedoch auch die Erzeugung hochfrequenter Signalanteile, die wegen der Diskretisierung des Signales zu Faltungsprodukten und damit zu Störungen führen, die auch nicht durch zusätzlichen Filteraufwand zu beheben sind.

Diese Argumente leiten dazu, die vom Band kommenden Signale, nur über den Kopfverstärker des Videokopfes verstärkt, direkt analog/digital umzusetzen und digital weiterzuverarbeiten. Da die vom Band kommenden FM-Signale mit starken AM-Störungen und Rauschen überlagert sind, muß somit ein FM-Demodulationsverfahren gewählt werden, das eine große AM-Unempfindlichkeit gewährleistet. Das im folgenden beschriebene digitale Verfahren, das prinzipiell einem analogen Ratiodetektor entspricht, zeigt eine hohe AM-Unempfindlichkeit. *Bild 4* zeigt die Prinzipschaltung des digitalen FM-Demodulators.

Das AM-behaftete FM-Eingangssignal $E = S \cdot \sin(\omega t)$ wird auf ein Transversal-Filter gegeben und gleichzeitig auf einen Speicher zum Laufzeitausgleich. Die Filtercharakteristik (hier ein $\sin^2$-Filter) ist die Diskriminatorkennlinie des FM-Demodulators und ist in *Bild 5* dargestellt. Das über diese Diskriminatorkennlinie „gelaufene“ Signal wird nun durch das um die Filterlaufzeit verzögerte Signal dividiert, und am Ausgang liegt dann ein von AM-Störungen befreites Signal vor, entsprechend den in Bild 4 eingetragenen Gleichungen, wobei S die Amplituden-modulierte Störung darstellt. Hierbei wird davon ausgegangen, daß die Änderung von S zwischen zwei Abtastwerten (Abtastfrequenz 17,73 MHz) klein ist.

Die Blockschaltung für das $\sin^2$-Filter zur Realisierung der Diskriminatorkennlinie ist in *Bild 6* dargestellt. Es ist ein sehr einfaches Transversalfilter, das über zwei Addierer und zwei Flipflops realisiert worden ist.

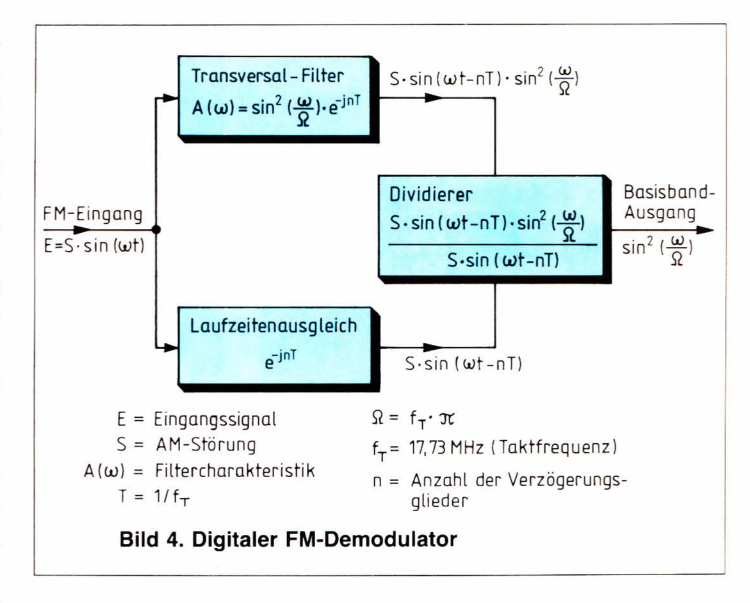


Bild 4. Digitaler FM-Demodulator

Dieser hier vorgestellte digitale FM-Demodulator benötigt jedoch einen digitalen Dividierer, d. h. einen Baustein, der in der Digitaltechnik bisher unüblich war. Da die Funktionsweise eines Pipelining-Dividierers schon ausführlich beschrieben worden ist [2], wird hier nicht weiter darauf eingegangen. Es soll nur noch einmal betont werden, daß die Realisierung dieses, mit einer Taktrate von 17,73 MHz arbeitenden digitalen Dividierers den Schlüssel für eine AM-unempfindliche FM-Demodulation darstellt.

Für sehr kleine Signale, d. h. für den Fall, daß der Zähler und der Nenner gegen Null gehen, wird dann noch eine Korrekturschaltung benötigt, um auch bei einer Division „Null“ durch „Null“ ein gültiges Ergebnis abliefern zu können.

Ein weiteres Verfahren, das zur FM-Demodulation eine Nulldurchgangsdetektion verwendet, zeigt ähnlich gute Ergebnisse und benötigt ebenfalls einen Dividierer.

In *Bild 7* ist schließlich die Blockschaltung der digitalen Video-Signalverarbeitung für die VCR-Aufnahme dargestellt und in *Bild 8* für die Wiedergabe. Vergleicht man diese mit Bild 2, so sieht man, daß hier strukturell

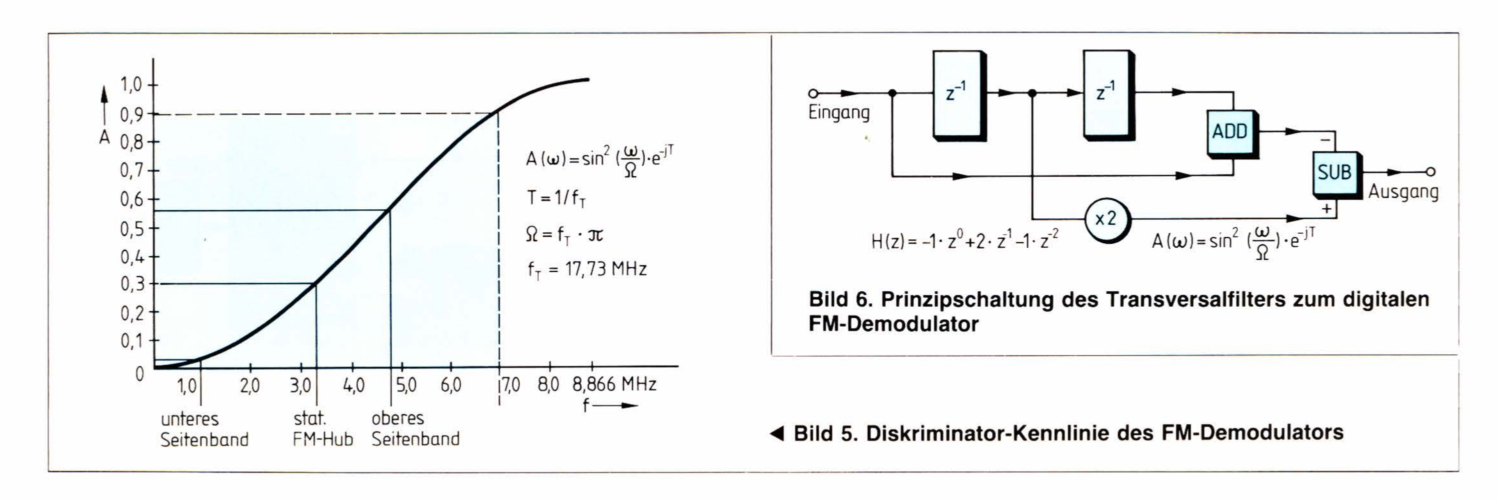


Bild 6. Prinzipschaltung des Transversalfilters zum digitalen FM-Demodulator

◀ **Bild 5. Diskriminator-Kennlinie des FM-Demodulators**

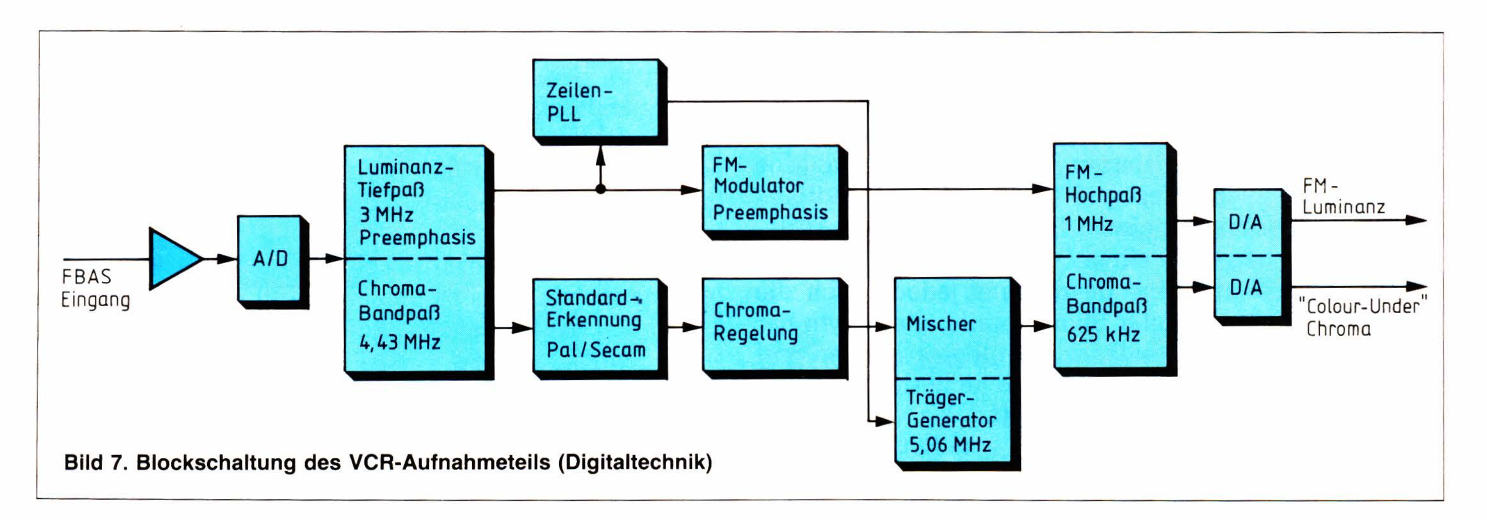


Bild 7. Blockschaltung des VCR-Aufnahmeteils (Digitaltechnik)

eine direkte Umsetzung der analogen Signalverarbeitung in die digitale Ebene vorliegt. Alle Funktionen, einschließlich der A/D- und D/A-Umsetzung, sind von ihren Geschwindigkeitsanforderungen in einem Standard-NMOS-Prozeß realisierbar und werden in einer ersten Stufe zu fünf ICs für die komplette Aufnahme und Wiedergabe führen. Hierbei ist anzumerken, daß man für die A/D-Umsetzung eine Auflösung von 7 Bit und für die D/A-Umsetzung eine Auflösung von 8 Bit benötigt.

4 System-Architektur

Das ganze System ist nach dem Verfahren eines „Distributed Processing" organisiert, bei dem einige anwendungsorientierte Echtzeit-Signalprozessoren, gesteuert von einem als Controller eingesetzten Universal-Mikrocomputer (z. B. MAB 84xx), überwacht und entsprechend beeinflußt werden. Zum Beispiel werden über den Mikrocomputer die einzelnen Filtercharakteristika innerhalb der Signalprozessoren dem jeweiligen Betriebszustand angepaßt. Man verfügt somit über eine hohe Systemflexibilität und kann individuelle Systemanpassungen im ROM-Speicher eines Mikrocomputers ablegen.

Der Autor wurde in Heft 3/1982 vorgestellt.

5 Digitalisierung erlaubt Zusatzfunktionen

Dieser erste Schritt einer Digitalisierung in Form einer „Eins zu Eins"-Umsetzung des analogen Systems ist einzig und allein auf eine Ökonomieverbesserung durch Peripheriereduktion orientiert. Allerdings kommen auch hier schon die phasenlinearen digitalen Filter einer Bildverbesserung entgegen. In weiteren Schritten ist es nun jedoch möglich, durch Zusatzschaltungen in dem digitalen System wesentliche Wiedergabeverbesserungen und auch einige Zusatzfunktionen zu realisieren.

Als Beispiele sollen hier eine Rauschreduktion und eine Zeitbasiskorrektur angeführt werden. Eine sehr effektive Rauschreduktion für die Wiedergabe ist bei geeigneter Wahl des Algorithmus mit sehr viel weniger als einem 1-MBit-Bildspeicher zu realisieren.

Eine Zeitbasiskorrektur, die die Ungleichmäßigkeiten im Bandlauf korrigiert, macht es möglich, daß man Signale von zwei unterschiedlichen VCR-Geräten mischen kann, was heute infolge der Zeitbasisfehler nicht möglich ist.

Literatur

[1] *Fischer, T.*: Fernsehen wird digital. ELEKTRONIK 1981, H. 16, S. 27...35.
[2] *Demmer, W., Draheim, P., Warmuth, L.* und *Lenth, J.*: Pipelining-Verfahren in der digitalen Signalverarbeitung. ELEKTRONIK 1982, H. 3, S. 73...77.

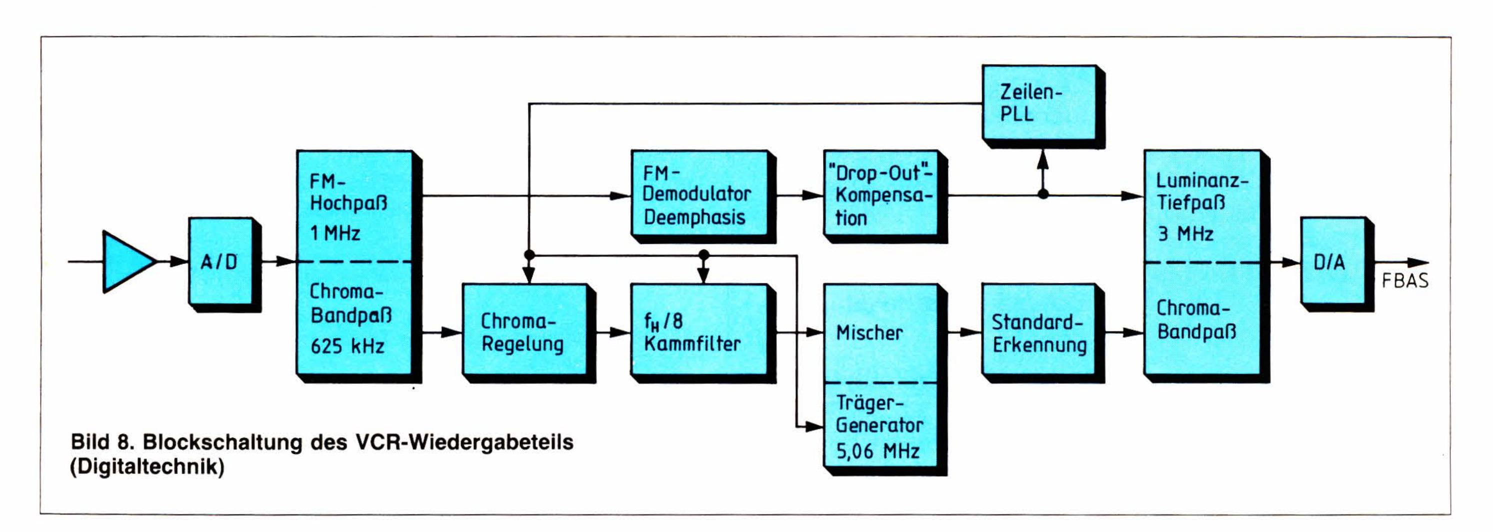


Bild 8. Blockschaltung des VCR-Wiedergabeteils (Digitaltechnik)

Dipl.-Ing. Albin Oberhofer

Zustandsregelung mit digitalen Filtern

Um die zahlreichen Erfahrungen im Umgang mit dem klassischen PID-Regler weiter anwenden zu können, wird mit digitalen Reglern meist eine quasi-stetige Realisierung der bekannten zeitkontinuierlichen Regler angestrebt. Die Abtastrate (Rechenschrittweite) muß dabei so festgelegt werden, daß der kontinuierliche Regler hinreichend genau simuliert wird. Die Dynamik der Regelstrecke und des Regelkreises geht dabei nicht unmittelbar in die Festlegung der benötigten Abtastrate ein. Dieser Beitrag zeigt, wie dieser Nachteil vermieden werden kann. Dazu wird ein diskretes Streckenmodell in Form einer z-Übertragungsfunktion ermittelt, mit dem dann direkt die z-Übertragungsfunktion des digitalen Reglers zu berechnen ist. Der Umweg über den Entwurf eines kontinuierlichen Reglers entfällt somit.

Zur Realisierung des Reglers können rekursive Digitalfilter verwendet werden. An einem abschließenden Beispiel wird deutlich, daß das auf der Zustandsregelung basierende Konzept der digitalen Regelung für beliebige Abtastraten anwendbar ist.

1 Konzept der Zustandsregelung

Ausgegangen wird von einer nicht sprungfähigen, linearen und zeitinvarianten Regelstrecke mit einem Stelleingang u(t) und $m \geq 1$ meßbaren Ausgängen $y_i(t)$, $i = 1,2,\ldots,m$, die im Ausgangsvektor $\underline{y}(t) = [y_1(t)\ y_2(t) \ldots y_m(t)]^T$ zusammengefaßt seien. Die Strecke ist mit dem Zustandsvektor $\underline{x}(t) = [x_1(t)\ x_2(t) \ldots x_n(t)]^T$ und den konstanten Matrizen $\underline{A}$, $\underline{b}$ und $\underline{C}$ durch die beiden Gleichungen

$$\dot{\underline{x}}(t) = \underline{A}\,\underline{x}(t) + \underline{b}\,u(t) \quad (1)$$

$$\underline{y}(t) = \underline{C}\,\underline{x}(t) \quad (2)$$

zu beschreiben [1].

Ein Zustandsregler erzeugt das Stellsignal u(t) mit dem Rückführvektor $\underline{k}^T = [k_1\ k_2 \ldots k_n]$ nach der Vorschrift

$$u(t) = \underline{k}^T \cdot \underline{x}(t) - L \cdot w(t) = \sum_{j=1}^{n} k_j \cdot x_j(t) - L \cdot w(t) \quad (3)$$

wobei nun die Führungsgröße w(t) das Eingangssignal des Zustandsregelkreises darstellt (*Bild 1*). Als Regelgröße kann ein beliebiges Streckenausgangssignal $y_i(t)$ festgelegt werden (mit einem Stellsignal ist nur ein Ausgangssignal frei zu regeln). Der Bewertungsfaktor L, mit dem das Führungssignal in den Regelkreis eingeführt wird, wird so festgelegt, daß stationär die Regelgröße den Wert der Führungsgröße annimmt.

Während die Dynamik der Regelstrecke durch die Nullstellen des charakteristischen Polynoms $\det(s\underline{I}-\underline{A})$ ($\underline{I}$: Einheitsmatrix) festliegt, kann die Dynamik des Regelkreises nach Bild 1 durch den Rückführvektor $\underline{k}$ beliebig beeinflußt werden, wenn die Strecke vollständig steuerbar ist [1].

Letzteres sei ebenso wie die vollständige Beobachtbarkeit im folgenden stets vorausgesetzt.

Durch geeignete Wahl von $\underline{k}$ können dem Zustandsregelkreis beliebige Eigenwerte – das sind die Nullstellen des charakteristischen Polynoms $\det(s\underline{I}-\underline{A}+\underline{b}\underline{k}^T)$ – aufgeprägt werden. Der Zweck der Anordnung nach Bild 1 ist die Beeinflussung der Streckendynamik durch beliebiges Verschieben der Streckeneigenwerte.

Das geschilderte Konzept ist offensichtlich nur dann realisierbar, wenn alle Zustandsgrößen meßtechnisch zugänglich sind. Dies ist aber in den meisten praktischen Anwendungen nicht der Fall, da oftmals die Kosten für die Installation der Meßglieder zu groß sind oder sie erst gar nicht angebracht werden können.

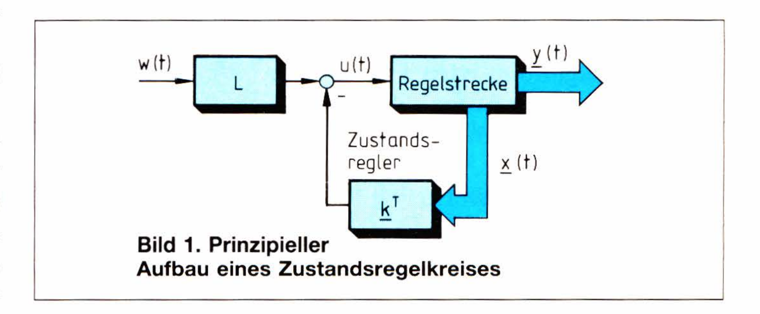


Bild 1. Prinzipieller Aufbau eines Zustandsregelkreises

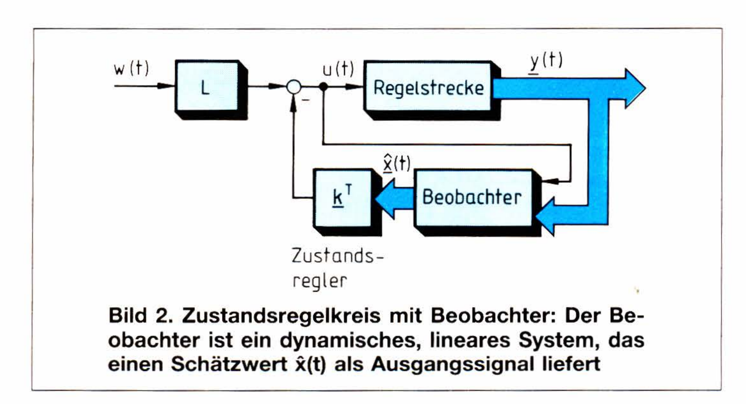


Bild 2. Zustandsregelkreis mit Beobachter: Der Beobachter ist ein dynamisches, lineares System, das einen Schätzwert $\hat{\underline{x}}(t)$ als Ausgangssignal liefert

Um auch in diesen praktisch relevanten Fällen das Konzept der Zustandsgrößen-Rückführung beibehalten zu können, ist ein Zustandsbeobachter einzusetzen, der einen Schätzwert $\hat{\underline{x}}(t)$ für den Streckenzustand bereitstellt. Der Regelkreis wird geschlossen, indem im Regelgesetz nach Gl. (3) anstelle von $\underline{x}(t)$ der Schätzwert $\hat{\underline{x}}(t)$ verwendet wird. Bemerkenswert ist, daß der Beobachter so konstruiert werden kann, daß das Regelkreisverhalten zwischen dem Führungseingang w(t) und dem Streckenausgang $\underline{y}(t)$ unabhängig davon ist, ob nun der beobachtete Zustandsvektor $\hat{\underline{x}}(t)$ oder der tatsächliche Streckenzustand $\underline{x}(t)$ zurückgeführt wird. Von dieser Tatsache wird an späterer Stelle Gebrauch gemacht werden.

Der Beobachter selbst ist ein dynamisches, lineares System mit m+1 Eingängen und n Ausgängen, welches bei Anregung mit den an der Regelstrecke anliegenden Signalen u(t) und $\underline{y}(t)$ die Komponenten $\hat{x}_i(t)$, i=1, ..., n, des Schätzwertes $\hat{\underline{x}}(t)$ liefert (*Bild 2*). Berücksichtigt man, daß zur Anwendung des Regelgesetzes nach Gl. (3) nur die Linearkombination $\underline{k}^T\hat{\underline{x}}(t)$ zugänglich sein muß, dann kann man auf die explizite Beobachtung des Streckenzustandes verzichten und den Zustandsregler mit dem Beobachter zu einer Einheit zusammenfassen. Damit stellt sich die Aufgabe, ein System mit m+1 Eingängen und einem Ausgang zur Bildung von $\underline{k}^T\hat{\underline{x}}(t)$ zu konstruieren. Es handelt sich offenbar um eine Filteraufgabe, die mit analogen Filtern unmittelbar behandelt werden kann. Im folgenden wird gezeigt, wie diese Aufgabe auch mit digitalen Filtern zu lösen ist, mit denen die Abtastwerte der an der Strecke anliegenden Signale individuell gefiltert werden.

2 Übergang zur Frequenzbereichsdarstellung

Die Realisierung des Zustandsregelungskonzeptes soll nur unter Verwendung der Signale u(t) und $\underline{y}(t)$ erfolgen, womit auch auf die Kenntnis der inneren Struktur der Regelstrecke, wie sie mit den Gl. (1), (2) beschrieben wird, verzichtet werden kann.

Es ist ausreichend, das Ein-/Ausgangsverhalten zwischen dem Stelleingriff u(t) und den Ausgängen $y_1(t)$, $y_2(t)$, ..., $y_m(t)$ zu kennen. Dies sei durch die Streckenübertragungsfunktionen

$$F_{S_i}(s) = \frac{L\{y_i(t)\}}{L\{u(t)\}}, \quad i = 1, 2, \ldots, m$$

bekannt und vorgegeben.

Der Einsatz von Digitalfiltern zur Regelung der kontinuierlichen Regelstrecke erfordert die Verwendung von Analog/Digital-Umsetzern zur Erfassung der Meßsignale und eines Digital/Analog-Umsetzers zur Erzeugung des analogen Stellsignals u(t). In *Bild 3* ist die gesamte Regelkreisstruktur dargestellt. Die Abtastung der Analogsignale erfolge im Abstand T (Abtastzeit) zu den Zeitpunkten t=kT (k=0,1,2,...). Der D/A-Umsetzer erzeuge ausgangsseitig aus der Folge u(kT) das kontinuierliche Stellsignal u(t) mit u(t) = u(kT) im Intervall kT≤t<(k+1)T.

Der nichtlineare Block in Bild 3 stellt einen Begrenzer dar, der zwei Aufgaben erfüllt. Zum einen wird damit sichergestellt, daß der D/A-Umsetzer nicht über den zulässigen Eingangszahlenbereich ausgesteuert wird. Die zweite Funktion ist die Nachbildung der grundsätzlich immer vorhandenen Stellsignalbegrenzung am Eingang der Regelstrecke (Stellglied); dadurch ist gesichert, daß dem Beobachter jenes (begrenzte) Stellsignal zugeführt wird, welches tatsächlich den Linearteil der Strecke erreicht. Der Beobachter ist somit auch dann in der Lage, den Streckenzustand zu rekonstruieren, wenn das Stellsignal in der Begrenzung ist. Diese für eine praktische Realisierung äußerst nützliche Struktur wurde erstmals von *Ch. Wurmthaler* angegeben [2, 3].

Die Aufgabe des Reglerentwurfs besteht im weiteren darin, die digitalen Filter in Bild 3 mit den Übertragungsfunktionen $G_{Ru}(z)$, $G_{R1}(z)$, $G_{R2}(z)$, ..., $G_{Rm}(z)$ so zu dimensionieren, daß der Regelkreis ein gewünschtes Verhalten

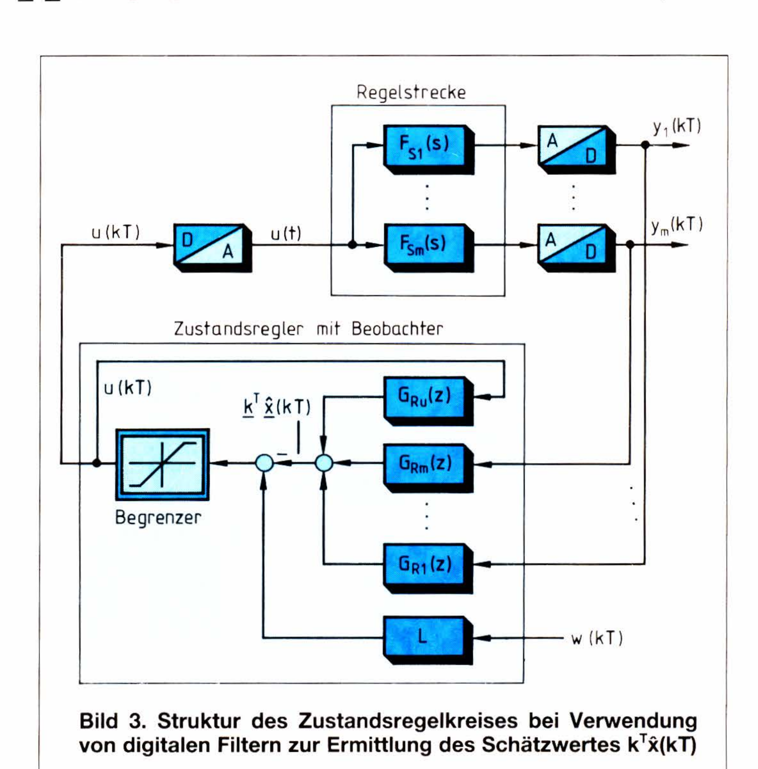


Bild 3. Struktur des Zustandsregelkreises bei Verwendung von digitalen Filtern zur Ermittlung des Schätzwertes $\underline{k}^T\hat{\underline{x}}(kT)$

aufweist. Bevor dies dargestellt werden kann, ist für die kontinuierliche Regelstrecke mit den angeschalteten D/A- und A/D-Umsetzern eine zeitdiskrete Beschreibung zu finden, so daß der gesamte Regelkreis ausschließlich mit diskreten z-Übertragungsfunktionen zu beschreiben ist.

3 Zeitdiskrete Beschreibung der Regelstrecke

In *Bild 4* ist ein Übertragungskanal der Regelstrecke mit den entsprechenden Umsetzern für sich dargestellt. Dieses Teilsystem wird eingangsseitig mit der Zahlenfolge u(kT) erregt und erzeugt daraufhin eine Ausgangsfolge y(kT). In [4] z. B. ist gezeigt, daß der erzeugende

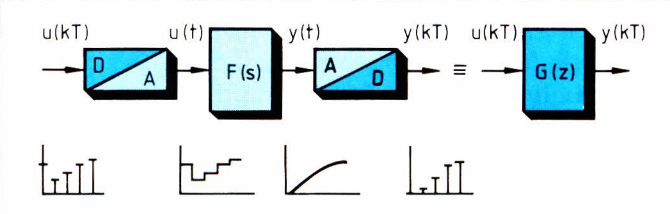


Bild 4. Anstelle des Analogfilters F(s) mit D/A- und A/D-Umsetzern kann das digitale Filter G(z) verwendet werden; die beiden Anordnungen sind identisch

Mechanismus zwischen u(kT) und y(kT) linear und damit durch eine diskrete Übertragungsfunktion beschreibbar ist. Ein digitales Filter G(z) als Streckenmodell hat genau dann das Verhalten der im Bild 4 gezeigten Anordnung, wenn die Vorschrift

$$G(z) = \frac{z-1}{z}\, Z\left\{ L^{-1}\left\{\frac{F(s)}{s}\right\}\Big|_{t=kT} \right\} \tag{4}$$

zur Ermittlung der Filterübertragungsfunktion G(z) angewendet wird [7].

Die Transformationsvorschrift nach Gl. (4) wird als sprungantwort-invariante Transformation bezeichnet, da die Sprungantwortfolge des Filters G(z) identisch ist mit der Folge der Abtastwerte der Sprungantwort des kontinuierlichen Systems F(s) zu den Zeitpunkten t = kT, k = 0,1,2... [5].

Zwischen den Polstellen der beiden Übertragungsfunktionen G(z) und F(s) aus Gl. (4) besteht der folgende einfache Zusammenhang: Hat F(s) einen Pol an der Stelle $s = s_\infty$, so hat G(z) einen Pol an der Stelle $z = e^{s_\infty \cdot T}$.

Wendet man Gl. (4) auf die m Streckenübertragungsfunktionen $F_{Si}(s)$ an, dann erhält man schließlich einen Satz von diskreten z-Übertragungsfunktionen

$$G_{Si}(z) = \frac{Z\{y_i(kT)\}}{Z\{u(kT)\}} = \frac{Z_{Si}(z)}{N_S(z)}$$

mit denen der Regelkreis auf die im *Bild 5* gezeigte Form zu bringen ist. Für die einzelnen Reglerübertragungsfunktionen $G_{Ru}(z)$, $G_{Ri}(z)$ sind die Zählerpolynome $Z_{Ru}(z)$, $Z_{Ri}(z)$ sowie das Nennerpolynom D(z) eingesetzt worden, wobei deren Grad zunächst noch nicht festgelegt ist.

4 Entwurf des digitalen Zustandsreglers

Die Entwurfsaufgabe besteht darin, die Übertragungsfunktionen in den einzelnen Pfaden des Reglers so zu bestimmen, daß gewisse Entwurfsvorgaben erfüllt sind.

Man ermittelt leicht aus Bild 5 für die Führungsübertragungsfunktionen den Ausdruck

$$G_{wi}(z) = \frac{Z\{y_i(kT)\}}{Z\{w(kT)\}} = \frac{L \cdot Z_{Si}(z) \cdot D(z)}{N_S(z) \cdot [Z_{Ru}(z) + D(z)] + \sum_{i=1}^{m} Z_{Si}(z) \cdot Z_{Ri}(z)} \tag{5}$$

Das frei vorgebbare Nennerpolynom D(z) der einzelnen Filter ist das charakteristische Polynom des eingebauten Zustandsbeobachters. Mit ihm ist festzulegen, in welcher Weise Beobachtungsfehler, die z. B. durch Störeingriffe entstanden sein können, abklingen. Zur Bestimmung der noch fehlenden Reglerpolynome $Z_{Ru}(z)$, $Z_{Ri}(z)$ wird nun von der eingangs erwähnten Möglichkeit

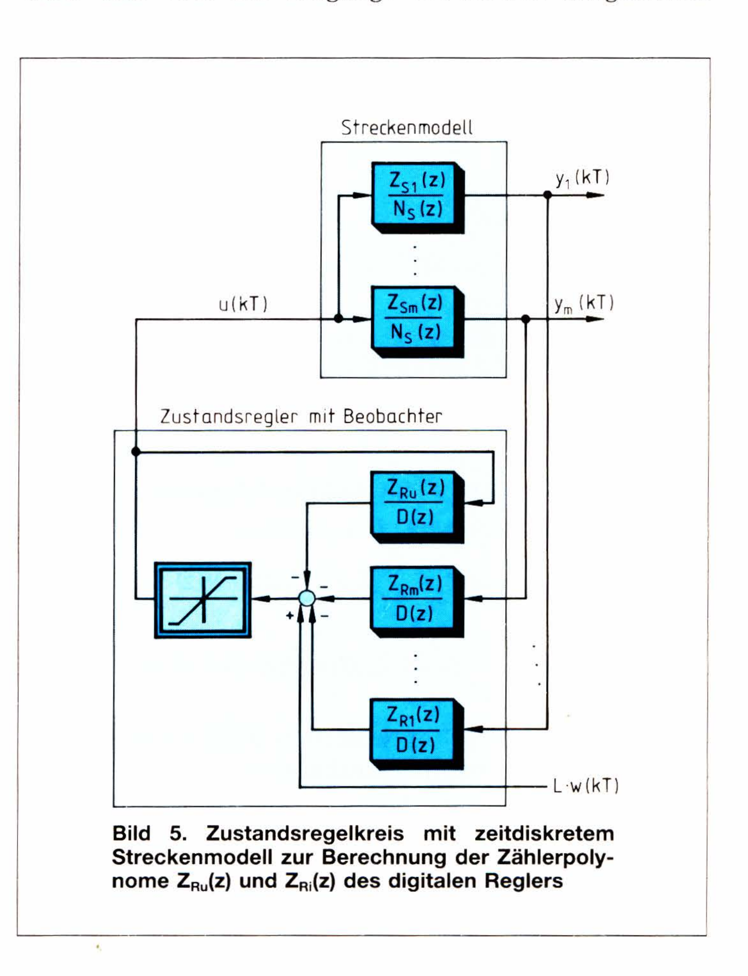


Bild 5. Zustandsregelkreis mit zeitdiskretem Streckenmodell zur Berechnung der Zählerpolynome $Z_{Ru}(z)$ und $Z_{Ri}(z)$ des digitalen Reglers

Gebrauch gemacht, das charakteristische Polynom des Regelkreises durch den Einsatz eines Zustandsreglers beliebig gestalten zu können.

Das charakteristische Polynom des hier vorliegenden Regelkreises ist in Gl. (5) im Nenner angegeben. Offenbar ist das Führungsverhalten genau dann vom Beobachterpolynom D(z) (und damit vom Beobachter selbst) unabhängig, wenn D(z) als Faktor im charakteristischen Polynom enthalten ist. Damit gelangt man zur Entwurfsglei-

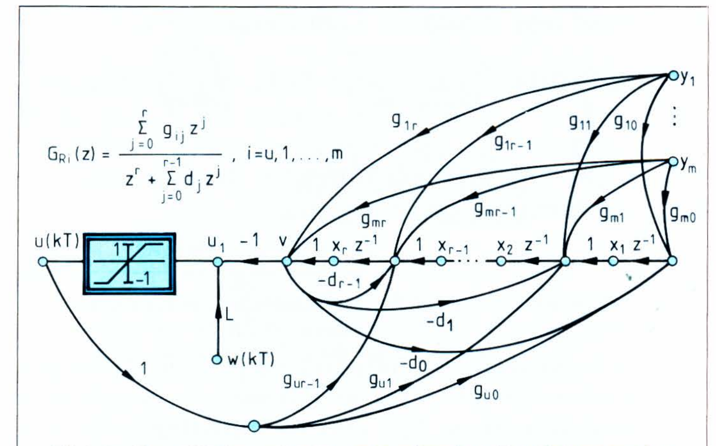


Bild 6. Signalflußgraph für eine günstige Realisierung des digitalen Zustandsreglers

```
100 REM             DIGITALER ZUSTANDSREGLER
110 REM         VOM GRAD R MIT M MESSGROESSEN
120 :
130 REM  HIER MESSGROESSEN Y(1),...,Y(M) ABTASTEN
140 :
150 V=X(R) : FOR I=1 TO M : V=V+G(I,R)*Y(I) : NEXT I
160 U1=L*W - V                          :REM BEGRENZEREINGANG
170 U=U1:IF  ABS(U1)>1  THEN U=SGN(U1)  :REM BEGRENZERAUSGANG
180 :
190 REM  HIER AUSGABE DES STELLSIGNALS U AN D/A-WANDLER
200 :
210 FOR J=R TO 1 STEP -1
220 X(J)=X(J-1) - D(J-1)*V + GU(J-1)*U
230 FOR I=1 TO M : X(J)=X(J) + G(I,J-1)*Y(I) : NEXT I
240 NEXT J
250 :
260 REM  WARTE BIS ABTASTZEIT VERGANGEN
270 :
280 GOTO 130                            :REM NAECHSTER ABTASTSCHRITT
```

Bild 7 BASIC-Programm zum Signalflußgraphen im Bild 6

Bild 7. BASIC-Programm zum Signalflußgraphen von Bild 6

chung, mit der die Reglerpolynome $Z_{Ru}(z)$, $Z_{Ri}(z)$ zu ermitteln sind:

$$N_S(z) \cdot [Z_{Ru}(z) + D(z)] + \sum_{i=1}^{m} Z_{Si}(z) \cdot Z_{Ri}(z) \overset{!}{=} \tilde{N}(z) \cdot D(z)$$

bzw.

$$N_S(z) \cdot Z_{Ru}(z) + \sum_{i=1}^{m} Z_{Si}(z) \cdot Z_{Ri}(z) \overset{!}{=} [\tilde{N}(z) - N_S(z)] \cdot D(z) \qquad (6)$$

Mit dem nach Gl. (6) entworfenen Regler ergeben sich die Führungsübertragungsfunktionen

$$G_{wi}(z) = \frac{L \cdot Z_{Si}(z)}{\tilde{N}(z)}$$

und man erkennt, daß im Vergleich mit den Übertragungsfunktionen

$$G_{Si}(z) = \frac{Z_{Si}(z)}{N_S(z)}$$

der ungeregelten Strecke das Nennerpolynom $N_S(z)$ durch das beliebig vorgebbare Polynom $\tilde{N}(z)$ ersetzt worden ist.

Zur Auswertung der Entwurfsgleichung (6) ist zu sagen, daß man zunächst die zu ermittelnden Reglerpolynome mit dem Grad $r = n - m$ allgemein ansetzt und durch Koeffizientenvergleich ein lineares Gleichungssystem zur Ermittlung der $(r+1) \cdot (m+1)$ unbekannten Polynomkoeffizienten gewinnt. Im Fall $m = 1$ ist das sich ergebende Gleichungssystem stets eindeutig lösbar. Ist $m > 1$, kann das Gleichungssystem unterbestimmt sein, was bedeutet, daß die gestellte Regelungsaufgabe mit einem Reglergrad $r < n - m$ gelöst werden kann.

5 Realisierung des digitalen Reglers

Die im Bild 3 dargestellte Struktur des Regelkreises kann unmittelbar als Realisierungsvorschrift betrachtet werden. Demnach sind $m+1$ rekursive Digitalfilter mit den Signalen $u(kT)$, $y_1(kT)$, ..., $y_m(kT)$ zu erregen und deren Ausgangssignale zu addieren. Offensichtlich ist diese Lösung nur dann praktikabel, wenn die Anzahl m der Meßsignale nicht zu groß ist.

Eine erhebliche Reduzierung des Aufwandes ergibt sich bei Berücksichtigung der Tatsache, daß sämtliche Reglerfilter das gleiche Nennerpolynom $D(z)$ haben. Dann nämlich sind die $m+1$ Teilfilter r-ten Grades durch ein einziges Filter vom Grad r mit $m+1$ Eingängen zu ersetzen.

In *Bild 6* ist der Signalflußgraph für diese Reglerrealisierung angegeben. Aus dem Signalflußgraphen kann neben einer Hardwarerealisierung auch ein Rechenalgorithmus für eine softwaretechnische Reglerrealisierung gewonnen werden. In *Bild 7* ist dieser Algorithmus in Form eines BASIC-Programms angegeben. Er eignet sich besonders zur Programmierung eines Signalprozessors.

Das nachfolgend angeführte Beispiel wurde mit einem entsprechend programmierten Bit-Slice-Rechenwerk (AMD 2903, Gleitkommaarithmetik, 16-Bit-Mantisse) erstellt. Dieses Gerät läßt sich z. B. bei Verwendung von $m = 2$ Meßsignalen als Regler vom Grad $r = 3$ mit einer Abtastfrequenz von 5 kHz betreiben [6]. Das ist für viele Regelungsprobleme ausreichend.

6 Beispiel

Zur Veranschaulichung des vorgestellten Konzeptes seien erzielte Ergebnisse bei der Regelung einer stark schwingungsfähigen Regelstrecke angegeben.

Die Regelstrecke mit der Übertragungsfunktion

$$F_S(s) = \frac{\omega_0^2}{s^2 + 2D\omega_0 s + \omega_0^2}, \quad \omega_0 = 1000\,s^{-1}, \; D = 0{,}2$$

wird in die Struktur nach Bild 3 eingebettet ($m = 1$).

Dipl.-Ing. Albin Oberhofer ist als Niederbayer in Vilsbiburg geboren. Er studierte Elektrotechnik zunächst an der Fachhochschule Nürnberg und dann an der Universität Erlangen-Nürnberg. Sein Studienschwerpunkt lag im Bereich der Regelungs- und Nachrichtentechnik. Seit zwei Jahren ist er als wissenschaftlicher Mitarbeiter am Institut für Regelungstechnik der Universität Erlangen-Nürnberg (Prof. Dr. H. Schlitt) tätig. Die digitale Regelung – insbesondere ihre Anwendung in der hydraulischen Antriebstechnik – ist sein Arbeitsbereich



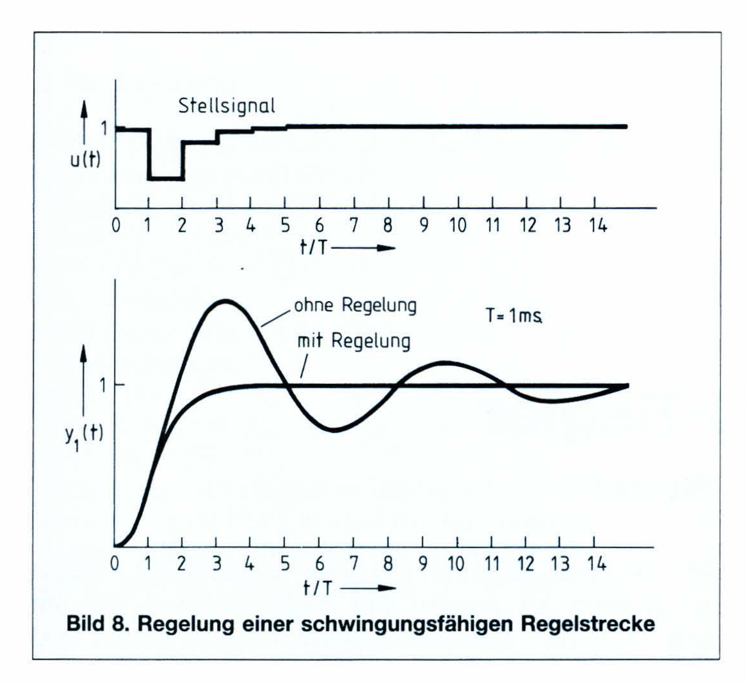


Bild 8. Regelung einer schwingungsfähigen Regelstrecke

Gefordert sei ein aperiodisch gedämpftes Übergangsverhalten zwischen dem Führungseingriff w und der Regelgröße (Streckenausgangssignal $y_1(t)$).

Mit der Abtastzeit T = 1 ms wird unter Verwendung von Gl. (4) zunächst die diskrete Streckenübertragungsfunktion

$$G_S(z) = \frac{0{,}405z + 0{,}353}{z^2 - 0{,}912z + 0{,}670}$$

ermittelt. Die gestellte Regelungsaufgabe ist mit Filtern ersten Grades lösbar. Aperiodisches Regelkreisverhalten ist z. B. mit der Vorgabe

$$\tilde{N}(z) = (z - 0{,}14)^2$$

$$D(z) = z - 0{,}2$$

(reelle Pole im Regelkreis) gewährleistet.

Die Lösung der Entwurfsgleichung (6) ermittelt eindeutig die Filterübertragungsfunktionen

$$G_{Ru}(z) = \frac{0{,}579}{z - 0{,}2}$$

$$G_{R1}(z) = \frac{0{,}132z - 0{,}730}{z - 0{,}2}$$

Der Führungsgrößenfaktor wird mit L = 0,976 so festgelegt, daß stationär die Regelgröße den Wert der Führungsgröße annimmt.

In *Bild 8* sind die Sprungantworten von geregelter und ungeregelter Strecke zusammen mit dem Stellsignal des digitalen Reglers angegeben.

7 Wahl der Abtastzeit

Mit dem dargestellten Konzept der Regelung mit digitalen Filtern ist für jede beliebige Wahl der Abtastzeit T gesichert, daß die Regelgröße zu den Zeitpunkten t = kT die mit der Entwurfsvorgabe ($\tilde{N}(z)$,D(z)) angestrebten Werte annimmt. Dabei ist zu beachten, daß der Verlauf der Regelgröße zwischen zwei aufeinanderfolgenden Abtastzeitpunkten nur vom Verhalten der Regelstrecke abhängt. Dies wird verursacht durch das treppenförmige Stellsignal, das nur zu den Abtastzeitpunkten vom Regler beeinflußt wird.

In *Bild 9* ist die im Beispiel verwendete Regelstrecke mit einer Abtastzeit von T = 20 ms geregelt. Die Führungssprungantwort hat zu den Abtastzeitpunkten den im Reglerentwurf eingebrachten Verlauf; die Interpolation dazwischen erfolgt mit der Sprungantwort der Strecke.

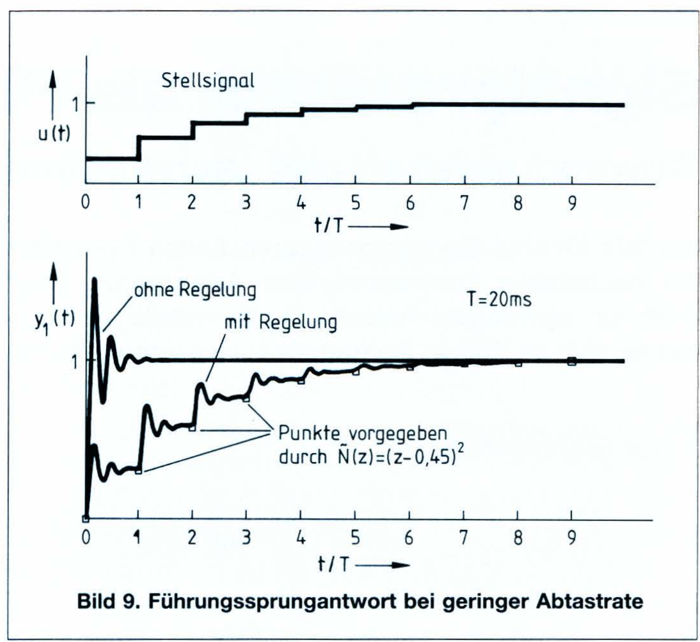


Bild 9. Führungssprungantwort bei geringer Abtastrate

Literatur

[1] *Unbehauen, R.:* Systemtheorie. R. Oldenbourg-Verlag, 1980.
[2] *Wurmthaler, Ch.:* Zustandsregler mit Störbeobachtern bei begrenzter Stellgröße. Regelungstechnik 28 (1980), S. 380...383.
[3] *Wurmthaler, Ch.:* Ein Beitrag zum Entwurf von Zustandsreglern und deren Einsatz unter praxisnahen Randbedingungen. Dissertation, Universität Erlangen-Nürnberg, 1979.
[4] *Ackermann, J.:* Abtastregelung, Band I, 2. Auflage, Springer-Verlag, 1983.
[5] *Stearns, S. D.:* Digitale Verarbeitung analoger Signale. R. Oldenbourg-Verlag, 1979.
[6] *Abraham, J.:* Implementierung und Erprobung eines Zustandsreglerprogramms auf einem mikroprogrammierbaren schnellen Rechenwerk. Diplomarbeit, Inst. f. Regelungstechnik der Universität Erlangen-Nürnberg, 1982.
[7] *Becker, H.-P.:* Entwurf und Realisierung digitaler Regler mit Mikroprozessoren. ELEKTRONIK 1984, H. 5, S. 51...56.

Ekkehard Kreß

Signalprozessor als PID-Regler

Anwendungsbeispiel „schwebende Kugel“

Digitale Einchip-Signalprozessoren finden bevorzugt als hochstabile Tonfrequenzfilter Anwendung. Aber auch für regelungstechnische Problemstellungen bieten sie sich an. Dieser Beitrag stellt eine Schaltung mit dem Baustein 2920 vor, die in der Lage ist, ein instabiles System zu regeln: Ein Elektromagnet läßt im beschriebenen Anwendungsbeispiel eine Eisenkugel frei in der Luft schweben.

1 Der Signalprozessor

Der Analog-Signalprozessor 2920 von Intel [1] ist in NMOS-Technologie hergestellt, verfügt über 192 × 24 Bit EPROM, 40 × 25 Bit RAM und ist in einem Gehäuse mit 28 Anschlüssen untergebracht. Der Baustein hat vier Analogeingänge und wahlweise acht Analog- bzw. Digitalausgänge (Auflösung: 9 Bit). Die Befehlszykluszeit beträgt 600 ns. Pro Zyklus können ALU-Befehle (ADD, SUB, LDA, XOR, AND, ABS, ABA, LIM), Shift- und Analogoperationen nebeneinander ausgeführt werden. Rechenoperationen werden mit 25 Bit Genauigkeit in der internen Darstellung im Bereich von +1 bis −1 abgearbeitet. Multiplikation und Division sind per Software durchzuführen. Für den Masseneinsatz ist eine ROM-Version mit 400 ns Zykluszeit verfügbar.

2 Versuchsaufbau

Als Demonstrationsmodell eines instabilen Systems eignet sich eine „schwebende Kugel“: Unter einem Elektromagneten mit vertikaler magnetischer Achse soll im Abstand von etwa 12 mm eine Eisenkugel im Durchmesser von 25 mm frei schwebend aufgehängt werden, deren Position mit einer Lichtschranke erfaßt wird. Der Magnetstrom soll derart gesteuert werden, daß die Kugelunterseite den Lichtstrahl der Schranke tangiert.

Die gestellte Aufgabe ist nur mit einer Regelung lösbar, da sich, wie aus der Erfahrung bekannt ist, bei festem Erregerstrom kein stabiler Zustand einstellt. Die Kugel wird bei kleinen Auslenkungen aus der labilen Gleichgewichtslage entweder an den Magneten herangezogen, oder sie fällt nach unten. Die Regelstrecke stellt ein instabiles System dar, da die Lageabhängigkeit der Hubkraft als Mitkopplung wirksam ist.

Durch Vorschaltung eines PID-Reglers mit geeignetem dynamischem Verhalten und eines Stellgliedes wird die Regelstrecke zu einem Lageregelkreis ergänzt. Dabei wird dem Reglereingang die Abweichung vom Sollwert über die Lichtschranke zugeführt. Der Regler erzeugt daraus eine Korrekturgröße, die über ein Stellglied den Erregerstrom steuert.

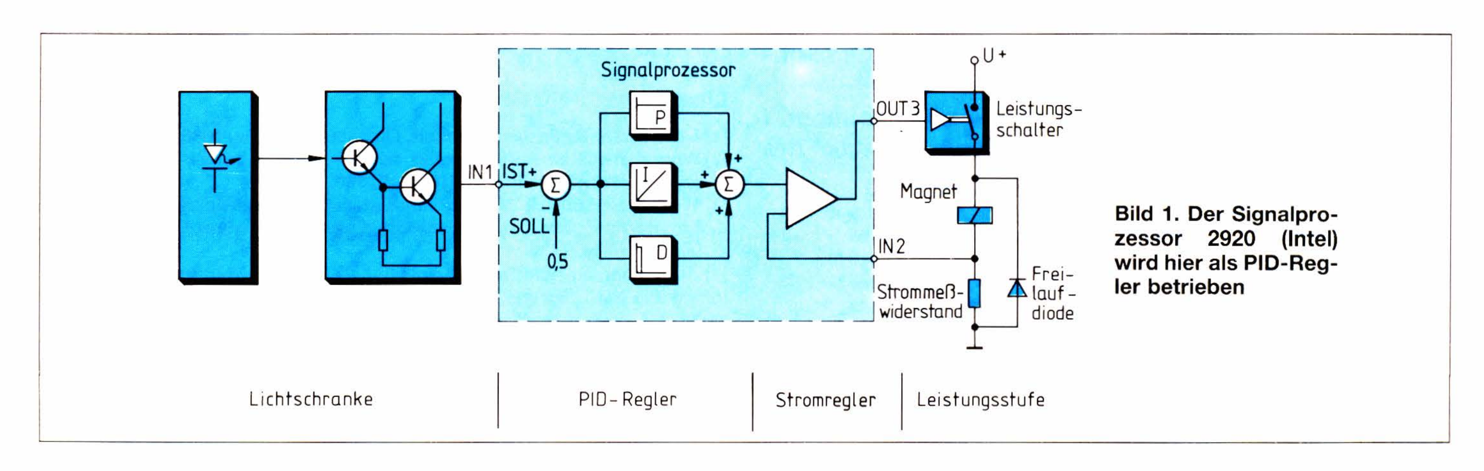


Bild 1. Der Signalprozessor 2920 (Intel) wird hier als PID-Regler betrieben

3 Realisierung

Das Demonstrationsmodell besteht aus folgenden Funktionsgruppen (*Bild 1*):
- Lichtschranke mit Emitterfolger zur Impedanzanpassung
- PID-Regler
- Stromregler
- Leistungsstufe mit Netzteil
- Elektromagnet

3.1 PID-Regler

Im Signalflußdiagramm des durch einen 2920 realisierten Reglers (*Bild 2*) sind die Komponenten des PID-Reglers, nämlich Proportional-, Integral- und Differentialteil gezeigt. Sein Ausgang wird zum Grundwert addiert und nach Skalierung als Sollwert dem Stromregler zugeführt. Der Grundwert ist eine eingefügte Konstante, die dem erforderlichen Strom für die Kugel-Sollposition entspricht. Die drei Reglerkomponenten erhalten als Eingangsgröße die Differenz aus Lichtschranken-Ausgangs-Spannung und Lichtschranken-spannungs-Sollwert.

Die Leerlaufspannung der Lichtschranke (freier Lichtstrahl) ist etwas höher als die Referenzspannung des 2920 eingestellt. Nach der Digitalisierung erhält somit die die Lichtschrankenspannung darstellende Programm-Variable bei Leerlauf den Wert 1,0, entsprechend dem Maximalwert des Bereichs.

Der Sollwert ist im Programm als Konstante mit dem Wert 0,5 definiert. Durch diese Wahl wird die Kugelunterseite etwa in der Mitte des Meßlichtstrahls, dessen wirksamer Durchmesser rund 1,5 mm beträgt, positioniert.

Der PID-Regler wird durch folgende Gleichung beschrieben [2]:

$$u_n = K_R \left[x_{dn} + \frac{T}{T_n} \sum_{v=0}^{n} x_{dv} + \frac{T_v}{T}(x_{dn} - x_{dn-1}) \right]$$
$$= \underbrace{K_R}_{Kp} x_{dn} + \underbrace{\frac{K_R}{T_n}}_{Ki} \cdot T \sum_{v=0}^{n} x_{dv} + \underbrace{K_R T_v}_{Kd} \frac{1}{T}(x_{dn} - x_{dn-1}) \quad (1)$$

$n = 0, 1, 2 \ldots$

u_n : Stellgröße
x_{dn} : Regelabweichung
T : Abtastperiode
T_n : Nachstellzeit
T_v : Vorhaltezeit
Kp : Koeffizient Proportionalanteil
Ki : Koeffizient Integralanteil
Kd : Koeffizient Differentialanteil

Die Koeffizienten Kp, Ki und Kd definieren die Sprungantwort des PID-Reglers und sind einfach und vor allem voneinander unabhängig zu programmieren. Hier zeigt sich ein großer Vorteil gegenüber analogen Lösungen, bei denen die Koeffizienten voneinander, je nach Schaltungsart mehr oder weniger, abhängig sind. In *Bild 3* ist die Sprungantwort des realisierten PID-Reglers dargestellt.

3.1.1 P-Anteil

Der Proportionalanteil ist durch eine Konstantenmultiplikation festgelegt. Hierzu muß die Konstante (Koeffizient Kp) in eine Summe von Zweierpotenzen aufgespaltet werden, um die Multiplikation durch Verschieben zu realisieren.

Beispiel:
$P := X \times Kp$; $Kp = 0{,}039 = 2^{-5} + 2^{-7}$
; X = Regelabweichung

Programmierung in 2920-Assembler:
```
LDA P,X,R5  ;P := X × 2^-5
ADD P,X,R7  ;P := P + X × 2^-7
```

3.1.2 I-Anteil

Beim Integralanteil wird pro Abtastperiode auf die Variable „I" (entspricht der Summe in Gleichung (1)) ein Wert addiert, der sich aus einer Konstantenmultiplikation mit der Regelabweichung „X" ergibt. Die Konstante ist gleich dem Produkt aus Abtastzeitintervall und dem Koeffizienten Ki und wird nachfolgend als Ki_T bezeichnet.

Beispiel:
$I := I + X \times Ki_T$; $Ki_T = 0{,}00244 = 2^{-9} + 2^{-11}$

2920-Assembler:
```
ADD I,X,R9   ; I := I + X × 2^-9
ADD I,X,R11  ; I := I + X × 2^-11
```

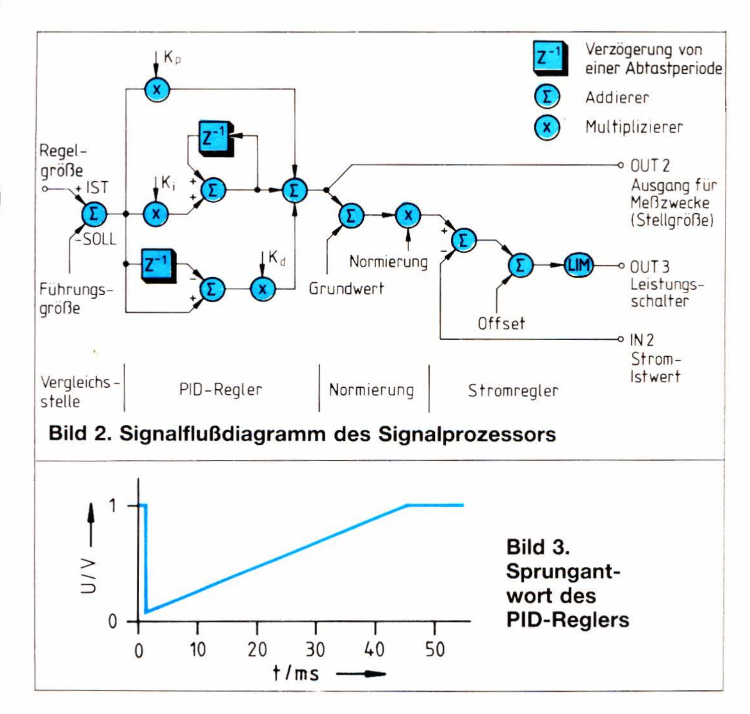
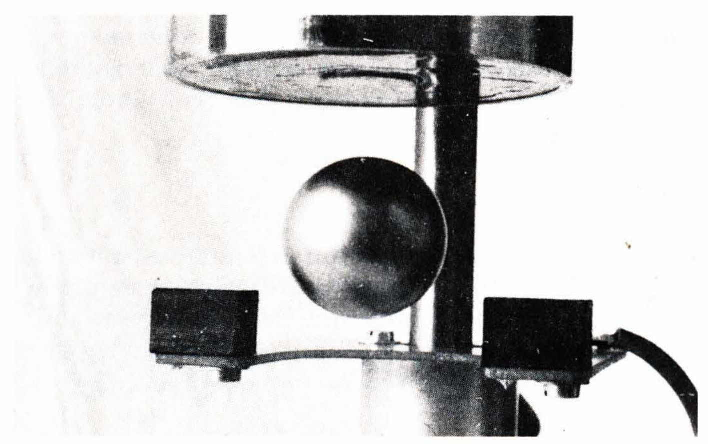


Bild 2. Signalflußdiagramm des Signalprozessors

Bild 3. Sprungantwort des PID-Reglers



Schwebende Kugel: ein anschauliches Beispiel für ein instabiles System

3.1.3 D-Anteil

Für den Differentialanteil wird im allgemeinen die Differenz aus der letzten und vorletzten Regelabweichung gebildet und mit der Konstanten Kd_T multipliziert. Die Konstante Kd_T errechnet sich als Produkt aus dem Koeffizienten Kd und dem Kehrwert des Abtastzeitintervalls.

Da bei dieser Anwendung die Abtastperiode (57,5 µs) sehr klein gegenüber der Systemreaktionszeit ist, wird die Differenz jeweils aus dem letzten und dem achtletzten Wert berechnet, um eine bessere Auflösung zu erhalten.

Beispiel:

$D := (X_n - X_{n-8}) \times Kd_T$; $Kd_T = 30$

2920-Assembler:

```
LDA X8,X7   ; X8 := X7
LDA X7,X6   ; X7 := X6
LDA X6,X5   ; X6 := X5
LDA X5,X4   ; X5 := X4
LDA X4,X3   ; X4 := X3
LDA X3,X2   ; X3 := X2
LDA X2,X1   ; X2 := X1
LDA X1,X    ; X1 := X
            .
            .
A/D-Konvertierungssequenz Lichtschrankenspannung
            .
            .
LDA X,DAR   ; X := LICHTSCHRANKENSPANNUNG
SUB X,KP4   ; X := X - 0,5
LDA D,X     ; D := LETZTE REGELABWEICHUNG
SUB, D,X8   ; D := D - X8

ADD D,D,L2  ; D := D × 5         (L2 = SHIFT 2 links)
ADD D,D,L1  ; D := D × 3         (L1 = SHIFT 1 links)
LDA D,D,L1  ; D := D × 2
; VARIABLE „D" ENTHÄLT D-ANTEIL
```

3.2 Stromregler

Sobald die Kugel von oben her den Lichtstrahl der Schranke unterbricht, wird der Ausgang des PID-Reglers gleich +1 gesetzt. Mit einer Skalierung wird der Variablen „SOLL" der Wert „PID" × 0,625, also 0,625 zugewiesen.

Am Eingang IN2 des 2920 ist der Erregerstrom-Meßwiderstand mit 0,1 Ω angeschlossen. Ein maximal zulässiger Magnetstrom von 6,25 A erzeugt die Spannung 0,625 V.

Da die Referenzspannung des Signalprozessors 1 V beträgt, entspricht dem maximalen Magnetstrom nach Analog/Digital-Umsetzung der Wert 0,625. Er wird im Programm der Variablen „I_IST" zugeordnet.

Der Komparator schaltet über seinen Ausgang die Leistungsstufe ab, falls Ist- und Sollwert gleich sind. Mit Hilfe der Skalierung wird damit eine Strombegrenzung von 6,25 A erreicht.

Die Komparatorfunktion läßt sich einfach mit einer Subtraktion realisieren. Eine maximale Verstärkung wird durch die Limes-Funktion des 2920 erreicht, dabei erhält die Zielvariable „M_ON" den „Full-Scale"-Wert mit dem Vorzeichen des Quelloperanden. „M_ON" kann also nur die Werte +1 oder −1 annehmen.

Da der LIM-Befehl eine Differenz von 0,0 als positiv betrachtet und damit dem Zieloperanden den Wert +1 zuweist, muß zum Differenzwert ein negativer Offset addiert werden, der kleiner als die Auflösung des A/D-Umsetzers ist. Damit wird verhindert, daß bei Gleichheit von Ist- und Sollwert der Magnetstrom eingeschaltet bleibt. Dies ist wichtig, falls die Stellgröße des PID-Reglers Null vorgibt. Ohne Korrektur kann der Erregerstrom nicht abgeschaltet werden.

Programmierung in 2920-Assembler:

Die Konstante 0,625 wird zerlegt in 0,5 und 0,125. Die Multiplikation mit 0,5 bzw. 0,125 ist mit einfachen Shiftoperationen durchzuführen. Hier ein Beispiel:

```
Beispiel:
LDA SOLL,PID,R1      ; SOLL := PID × 0,5                R1 = SHIFT 1 rechts
ADD SOLL,PID,R3      ; SOLL := SOLL + PID × 0,125       R3 = SHIFT 3 rechts

SUB SOLL,I_IST       ; STROMKOMPARATOR
SUB SOLL,KP1,R7      ; NEGATIVER OFFSET FÜR LIM (0,000976563)
LIM M_ON,SOLL        ; Limes Funktion
; Es folgt Ausgabesequenz für die Variable M_ON
```

3.3 Leistungsstufe

Die Leistungsstufe (*Bild 4*) arbeitet als Schaltregler mit hohem Wirkungsgrad. Die Steuerspannung vom Signalprozessor an OUT3 (Variable M_ON) beträgt +1 V für Magnetstrom ein und −1 V für aus.

Wird der Leistungstransistor abgeschaltet, bricht das Magnetfeld zusammen, der Selbstinduktionsstrom fließt über Freilaufdiode und Meßwiderstand. Der Komparator im 2920 erkennt, daß der Magnetstrom abnimmt und schaltet den Magneten wieder an die Versorgungsspannung. Auf diese Weise wird der Schalttransistor permanent wechselweise angesteuert. Dem Magnetstrom ist nur eine geringe Welligkeit überlagert, die hauptsächlich von der Erregerwicklungsinduktivität abhängig ist. Die Schaltfrequenz ist durch die Abtastrate, bedingt durch die Taktfrequenz und die Programmlänge, gegeben.

Der mittlere Erregerstrom ist eine Funktion des Verhältnisses der Einschaltzeit zur Ausschaltzeit, der Versorgungsspannung und des Magnetwiderstandes.

Für eine möglichst einfache und zuverlässige Erregerstromerfassung durch den 2920 muß der Meßwiderstand auf Massepotential liegen. Dies bedingt, daß der Schalt-

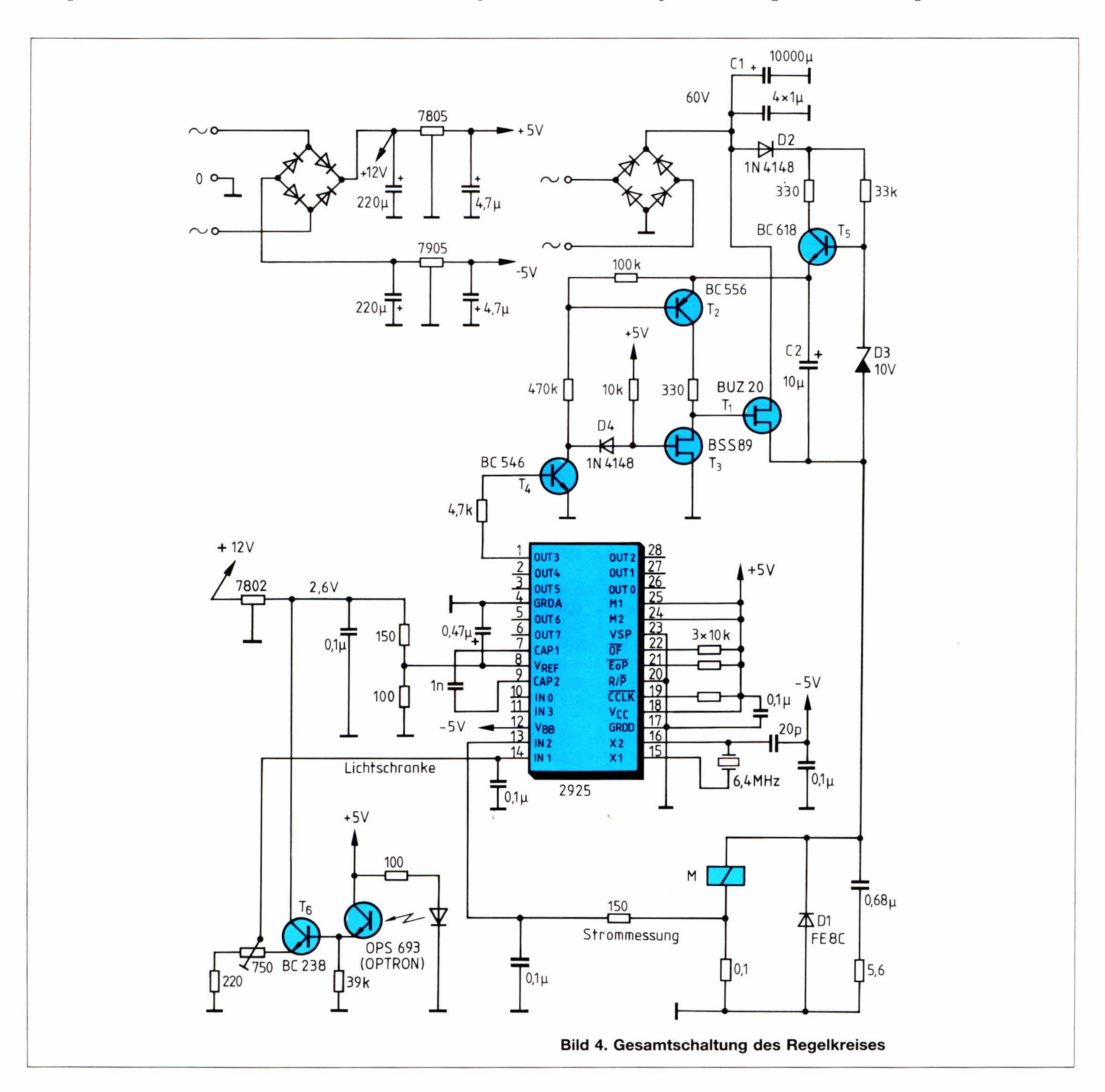


Bild 4. Gesamtschaltung des Regelkreises

transistor direkt an der Plusseite der Spannungsversorgung angeschlossen sein muß. Für diese Schaltungsart ist normalerweise bei bipolaren Transistoren eine eigene Hilfsspannungsquelle erforderlich.

Der verwendete VMOS-Leistungstransistor benötigt im statischen Betrieb keine Steuerleistung. Lediglich beim Ein- bzw. Ausschalten muß die Gatekapazität umgeladen werden. Für schnelle Schaltzeiten und damit hohen Wirkungsgrad ist deshalb eine niederohmige Gateansteuerung wichtig.

Die Hilfsspannungsquelle ist mit dem Kondensator C_2 realisiert, der jeweils in der Sperrphase des Schalttransistors aus der Versorgungsspannung über Transistor T5 nachgeladen wird. Mit der Z-Diode im Basiskreis von T5 wird eine Begrenzung der Hilfsspannung erreicht. Um die Leistungsstufe einzuschalten, wird T2 durch T4

```
ISIS-II 2920 ASSEMBLER V1.0

ASSEMBLER INVOKED BY: AS2920 KUGEL.ASM

  LINE  LOC OBJECT SOURCE STATEMENT

     1              $PAGELENGTH(72) DEBUG
     2              $TITLE('SCHWEBENDE KUGEL gesteuert mit "2920" *V1.3*  08.04.83')
     3              ;
     4              ;      ***************************************************
     5              ;      *                                                 *
     6              ;      *                SCHWEBENDE KUGEL                 *
     7              ;      *                                                 *
     8              ;      ***************************************************
     9              ;
    10              ;      (C) E.KRESS 83
    11              ;
    12              ; BESCHREIBUNG:
    13              ;
    14              ;      UNTER EINEM ELEKTROMAGNETEN SCHWEBT EINE EISENKUGEL,
    15              ;      DEREN POSITION MIT EINER LICHTSCHRANKE ERFASST WIRD.
    16              ;
    17              ;      DER SIGNALPROZESSOR "2920" REGELT DEN MAGNETSTROM
    18              ;      DERART, DASS DIE KUGEL DIE LICHTSCHRANKE TANGIERT.
    19              ;
    20              ;      HIERZU IST IM "2920" EIN PID-REGLER MIT NACHFOLGENDEN
    21              ;      ZWEIPUNKT-REGLER REALISIERT.
    22              ;
    23              ;      DEM PID-REGLER WIRD DIE DIFFERENZ AUS LICHTSCHRANKEN-
    24              ;      SOLLWERT (MITTENPOSITION) UND ISTWERT (EINGANG IN1)
    25              ;      ZUGEFUEHRT.  DER REGLERAUSGANG WIRD MIT EINEM GRUND-
    26              ;      WERT ADDIERT, SKALIERT UND ALS STROMSOLLWERT DEM ZWEI-
    27              ;      PUNKT STROMREGLER GEGEBEN.
    28              ;      DER STROMISTWERT WIRD AN EINEM MESSWIDWERSTAND ALS
    29              ;      SPANNUNG ABGENOMMEN UND DEM EINGANG IN2 ZUGEFUEHRT.
    30              ;
    31              ;      DER ZWEIPUNKT-REGLER VERGLEICHT SOLL- UND ISTWERT.
    32              ;      STEIGT DER ISTWERT UEBER DEN SOLLWERT, SO WIRD DER
    33              ;      MAGNETSTROM MITTELS DER LEISTUNGSSTUFE ABGESCHALTET,
    34              ;      BZW. WIEDER EINGESCHALTET WENN DER ISTWERT UNTER DEN
    35              ;      SOLLWERT SINKT.
    36              ;      DIE LEISTUNGSSTUFE ARBEITET IM SCHALTBETRIEB UND WIRD
    37              ;      VOM AUSGANG OUT3 MIT DEM MAXIMALEN POSITIVEN BZW.
    38              ;      NEGATIVEN PEGEL ANGESTEUERT.
    39              ;
    40              ;      EINE GENAUE BESCHREIBUNG IST IM
    41              ;          ANWENDUNGSBERICHT 2920  "SCHWEBENDE KUGEL"
    42              ;      ENTHALTEN.
    43              ;                     ***************
    44              ;
    45              ; CLOCK FREQUENZ      :   6.4 MHz
    46              ;
    47              ; REFERENZSPANNUNG    :   1.0 V
    48              ;
    49              ; ABTASTPERIODE       :   92 * 0,625us = 57,5us
    50              ;
    51              ; BENUTZTE EIN- UND AUS-GAENGE:
    52              ;
    53              ;      IN1  = LICHTSCHRANKE
    54              ;      IN2  = SPULENSTROM
    55              ;
    56              ;      OUT3 = ANSTEUERUNG SCHALTTRANSISTOR
    57              $EJECT
    58     0 B000EF     OUT3
    59     1 B000EF     OUT3
    60     2 B000EF     OUT3
    61     3 B000EF     OUT3
    62
    63              ; ** KANAL 1 DIGITALISIEREN (LICHTSCHRANKE) **
    64
    65     4 1000EF     IN1
    66     5 1000EF     IN1
    67     6 1000EF     IN1
    68     7 1000EF     IN1
    69     8 1000EF     IN1
    70     9 1000EF     IN1
    71    10 1000EF     IN1
    72    11 10C6EF     LDA DAR,KP0,IN1
    73    12 4000EF     NOP
    74    13 6000EF     CVTS
    75    14 4282EF     LDA GW,KP2
    76    15 408AAC     ADD GW,KP1,R6
    77    16 7100EF     CVT7
    78    17 428A8A     SUB GW,KP3,R5
    79    18 4000EF     NOP
    80    19 6100EF     CVT6
    81    20 4000EF     NOP
    82    21 4000EF     NOP
    83    22 5100EF     CVT5
    84    23 4000EF     NOP
    85    24 4000EF     NOP
    86    25 4100EF     CVT4
    87    26 4200FF     LDA LS8,LS7
    88    27 4608EF     LDA LS7,LS6
    89    28 3D00FF     LDA LS6,LS5,CVT3
    90    29 4818EF     LDA LS5,LS4
    91    30 4A10FF     LDA LS4,LS3
    92    31 2100EF     CVT2
    93    32 4E18EF     LDA LS3,LS2
    94    33 4430FF     LDA LS2,LS1
    95    34 1100EF     CVT1
    96    35 48C2FF     LDA LS_D,KP4
    97    36 0360EF     LDA LS1,LS0,CVT0
    98
    99              ; ** MITTELWERT LICHTSCHRANKENSPANNUNG **
   100
   101    37 24621E     LDA LS,DAR,R1,IN2
   102    38 2C601C     ADD LS,LS_V,R1,IN2
   103    39 227BEF     LDA LS_V,LS,IN2
   104
   105    40 266BEF     LDA LS0,LS,IN2
   106
   107              ; ** REGELABWEICHUNG **
   108
   109    41 226BFB     SUB LS_D,LS,IN2
   110
```

vom Signalprozessor aus leitend gesteuert. Gleichzeitig zieht T4 das Gatepotential von T3 über die Diode D4 auf Masse (T3 sperrt). Transistor T2 schaltet die Hilfsspannung an das Gate des Leistungstransistors T1, und dieser wird leitend.

Wechselt die Ausgangsspannung des 2920 von +1 V nach −1 V, werden T4 und T2 gesperrt. Die Gatespannung von T3 ist nicht mehr nach Masse geklemmt. T3 ist leitend, die Ladung der Gatekapazität von T1 fließt über T3 nach Masse ab. Der Schalttransistor T1 sperrt. Das vollständige Programm zeigt *Bild 5*.

4 Software-Entwicklung

Die Software kann entweder sehr preiswert mit dem „2920 System Design Kit“ [3] oder mit einem Entwick-

```
111                   ; **** KANAL 2 (STROMWERT) DIGITALISIEREN ****
112
113   42 2066EB    SUB DAR,DAR,IN2
114
115                   ; ** D - REGLER **
116
117   43 2270FF    LDA DIFF,LS0,IN2
118   44 2058FB    SUB DIFF,LS8,IN2 ; KUGEL AB ==> SIGN=NEG
119
120   45 4878BD    ADD DIFF,DIFF,L2
121   46 6878DD    ADD DIFF,DIFF,L1,CVTS ; DIFF := DIFF * 15
122
123                   ; ** I - REGLER **
124
125   47 44780D    ADD INT,LS_D,R9
126   48 44784D    ADD INT,LS_D,R11        ; KUGEL AB ==> SIGN=POS
127
128                   ; ** P - REGLER **
129
130   49 75789E    LDA PROP,LS_D,R5,CVT7
131   50 4478DC    ADD PROP,LS_D,R7
132
133                   ; ** SUMME: P-I-D **
134
135   51 4A21EF    LDA PID,INT
136   52 6B29ED    ADD PID,PROP,CVT6
137   53 4829EB    SUB PID,DIFF
138   54 4829EB    SUB PID,DIFF
139
140                   ; ** STROMGRUNDWERT MIT PID ADDIEREN **
141
142   55 5101FF    LDA GWPID,GW,CVT5
143   56 4081FD    ADD GWPID,PID
144
145                   ; ** STROMSOLLWERT (I-SOLL-MAX = 6,25) **
146
147   57 44890E    LDA SOLL,GWPID,R1
148   58 45894C    ADD SOLL,GWPID,R3,CVT4 ; SKALIERUNG
149
150   59 4000EF    NOP
151   60 4000EF    NOP
152   61 3100EF    CVT3
153   62 4000EF    NOP
154   63 4000EF    NOP
155   64 2100EF    CVT2
156   65 4000EF    NOP
157   66 4000EF    NOP
158   67 1100EF    CVT1
159   68 4000EF    NOP
160   69 4000EF    NOP
161   70 0100EF    CVT0    ; STROMWERT IN "DAR" IST POSITIV
162   71 4423FF    LDA I_IST,DAR
163
164                   ; *** KONTROLLAUSGABE PID ***
165
166   72 40C4EF    LDA DAR,PID
167   73 4000EF    NOP
168   74 4000EF    NOP
169   75 4000EF    NOP
170   76 A000EF    OUT2
171   77 A000EF    OUT2
172   78 A000EF    OUT2
173   79 A000EF    OUT2
174   80 A000EF    OUT2
175   81 A000EF    OUT2
176   82 A000EF    OUT2
177   83 A000EF    OUT2
178
179                   ; ** STROMREGLER **
180
181   84 4689EB    SUB SOLL,I_IST
182   85 448BCA    SUB SOLL,KP1,R7 ; NEGATIVER OFFSET FUER LIM
183
184                   ; ** SCHALTTRANSISTOR ANSTEUERUNG **
185
186   86 42C4E5    LIM DAR,SOLL
187   87 4000EF    NOP
188   88 5000EF    EOP
189   89 4000EF    NOP
190   90 B000EF    OUT3
191   91 B000EF    OUT3
192                END

SYMBOL:                   VALUE:

GW                          0
LS8                         1
LS7                         2
LS6                         3
LS5                         4
LS4                         5
LS3                         6
LS2                         7
LS1                         8
LS_D                        9
LS0                        10
LS                         11
LS_V                       12
DIFF                       13
INT                        14
PROP                       15
PID                        16
GWPID                      17
SOLL                       18
I_IST                      19

ASSEMBLY COMPLETE
ERRORS    =   0
WARNINGS  =   0
RAMSIZE   =  20
ROMSIZE   =  92
```

Bild 5. Dieses Programm macht aus dem Signalprozessor einen PID-Regler



Ekkehard Kreß ist in Wasserburg/Inn geboren. Er absolvierte eine Lehre als Elektromechaniker und studierte anschließend Informatik an der Fachhochschule München. Zur Zeit befindet er sich im Abschlußsemester. Schon seit 1975 betätigte er sich als freiberuflicher Entwickler für Schaltnetzteile.
Hobbys: RC-Cars mit Verbrennungsmotor, Fotografieren.

lungssystem unter dem Betriebssystem „ISIS II" erstellt werden.

Das „Design Kit" verfügt über einen Zeilenassembler. Die Programme können über eine Audiobuchse auf Kassette abgelegt bzw. von dort geladen werden. Das gleiche gilt für die vorhandene V.24-Schnittstelle in Verbindung mit einem Entwicklungssystem. Mit dem auf der Platine befindlichen Programmiersockel wird das EPROM des 2920 programmiert.

Mit dem „2920 Support Package" ist unter ISIS II eine komfortable Softwareentwicklung gegeben: Das Quellprogramm wird mit dem „2920-Assembler" [4] übersetzt. Danach besteht mit dem „2920-Simulator" [5] die Möglichkeit, das Programm auszutesten. Hierfür können analoge Eingangsfunktionen (SIN, COS, EXP, LOG, usw.) definiert werden. Alle internen Register, Programmspeicher, Eingangs-, Ausgangswerte und andere Systemvariablen des 2920 können überprüft und modifiziert werden. Der interne Ablauf des Signalprozessors wird simuliert. „Trace" und bedingte „Breakpoints" sind möglich.

Das ausgetestete und gegebenenfalls modifizierte Objektmodul wird dann mit dem UPP (*Universal PROM Programmer*) in den 2920 übertragen.

5 Anmerkungen

Bei der gezeigten Anwendung wird nur die Hälfte des verfügbaren Programmspeichers gebraucht und die volle Leistungsfähigkeit des Bausteins nicht genutzt. Die eigentliche Reglercodierung benötigt etwa ein Viertel des Programms. Die restlichen Programmteile sind für die Ein- und Ausgabesequenzen, zur Variableninitialisierung und zum Aufbau von Konstanten notwendig. Für aufwendigere Kaskadenregler oder für Eingangsgrößenfilterung wäre noch hinreichend Platz vorhanden.

Literatur

[1] 2920 Analog Signal Prozessor Design Handbook. Fa. Intel, 1980.
[2] *Föllinger, D.*: Regelungstechnik. 2. Auflage, Elitera-Verlag.
[3] SDK-2920 System Design Kit User's Guide. Fa. Intel, 1981.
[4] 2920 Assembly Language Manual. Fa. Intel, 1979.
[5] 2920 Simulator User's Guide. Fa. Intel, 1980.

Dipl.-Ing. Hans Volkers

Schnelle Fouriertransformation mit TMS320 als Coprozessor

Dieser Beitrag beschreibt den Einsatz des Signalprozessors TMS320 als frei programmierbaren Coprozessor im Verbund mit einem vorhandenen Host-Rechner. Dabei legt der Autor besonderen Wert auf die Darstellung der Rechnerkopplung und die Behandlung der spezifischen Probleme einer schnellen Fouriertransformation (FFT). Die Kopplung nimmt ein umschaltbarer Speicher vor, der im realisierten Aufbau den Anschluß mehrerer TMS320 sowie parallele Aktivitäten aller Rechner zuläßt.

1 TMS320 als Coprozessor

1.1 *Einführung*

Mit einer Verarbeitungsgeschwindigkeit von 5 Mio. Instruktionen/s sowie einem integrierten Hardware-Multiplizierer liegt der Datendurchsatz des Signalprozessors TMS320 bei FFT-Algorithmen mit 16-Bit-Festkommaarithmetik im Mittel um den Faktor 20 höher als bei einer CPU 8086 (8 MHz). Eine Kombination von 8086 und 8087 würde nur bei Gleitkommaarithmetik entscheidende Geschwindigkeitsvorteile bringen.

Da die zu verarbeitenden Daten in den meisten Fällen von einem Analog-/Digital-Umsetzer übernommen werden, stellt die 16-Bit-Festkommaarithmetik bei den genannten Algorithmen keine Einschränkung der Rechengenauigkeit dar.

Der Einsatz des TMS320 als Coprozessor an einem µP-System (im folgenden Host-µP genannt) ermöglicht eine sehr hohe Verarbeitungsgeschwindigkeit des Gesamtsystems. Hierzu wird der TMS320 als frei programmierbarer Coprozessor die Programmteile übernehmen, für die er optimiert ist. Zu den Anwendungsfällen zählen:
- Transformationen
- Korrelationen
- Matrizenverarbeitung
- Berechnung nichtlinearer Funktionen.

1.2 *Prinzip der Rechnerkopplung*

Das wesentliche Hardwareproblem des Multiprozessorkonzepts ist die Art der Rechnerkopplung. Sie muß die folgenden Möglichkeiten zur Verfügung stellen:

– Programmierbarkeit der Coprozessoren

Der Host-µP stellt mit seinen Ein-/Ausgabemöglichkeiten wie Bildschirm, Massenspeicher und der verfügbaren Software den koordinierenden und übergeordneten Systemteil dar. Er sollte deshalb je nach aktueller Problemstellung die Programme für die Coprozessoren laden können.

– Datenaustausch zwischen den Prozessoren

In der Praxis genügt ein Austausch von Daten zwischen Host-µP und den einzelnen Coprozessoren (*Bild 1*). Die Realisierung einer vollständigen konfliktfreien

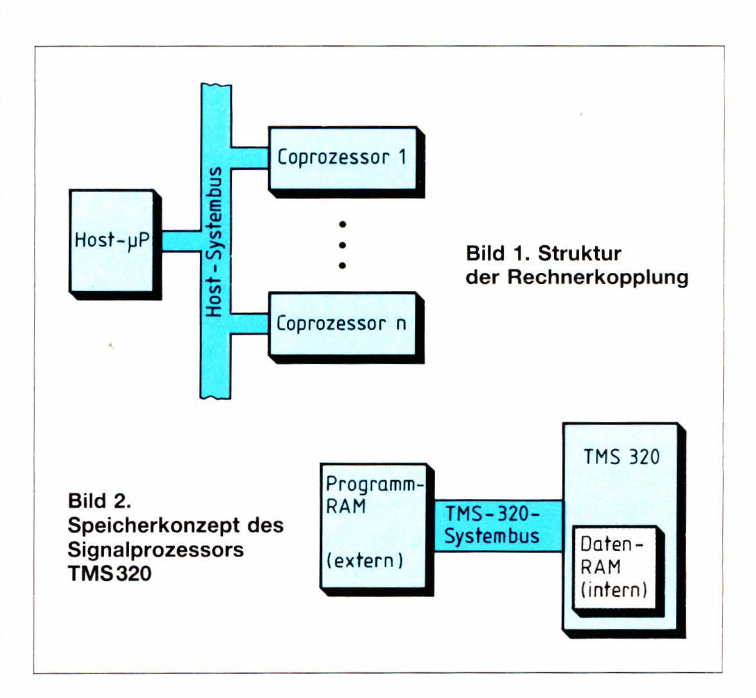


Bild 1. Struktur der Rechnerkopplung

Bild 2. Speicherkonzept des Signalprozessors TMS320

Vernetzung der Rechner ist zu aufwendig. Der Datenaustausch kann über die E/A-Ports oder den Programmspeicher des TMS 320 erfolgen. Im folgenden wird die Kopplung über einen gemeinsamen Speicher für Host-µP und TMS 320 beschrieben.

Zum Verständnis des Datenaustausches muß auf das Prozessorkonzept des TMS 320 eingegangen werden. Er besitzt eine modifizierte Harvard-Architektur, d.h. eine fast vollständige Trennung zwischen Programm- und Datenspeicher (*Bild 2*). Die Modifizierung erlaubt mit den „TABLE READ"- und „TABLE WRITE"-Befehlen (TBLR/TBLW) das Kopieren von Werten aus dem Programm- in den Datenspeicher bzw. umgekehrt. Diese Instruktionen arbeiten zweifach indiziert, die jeweiligen Zieladressen stehen im Akkumulator und in einem Register, die Ausführungszeit beträgt 600 ns.

Bild 3. Umschaltspeicher zur Rechnerkopplung

Bild 4. Typischer Programmablauf

Zur Übertragung von Daten und Programm ist somit der Zugriff des Host-µP auf den externen Programmspeicher des TMS 320 hinreichend. Die Forderung nach Programmierbarkeit des Coprozessors ist damit automatisch gelöst. Eine mögliche Realisierung der Kopplung wäre der Aufbau eines Dual-Port-RAM für den Programmspeicher des TMS 320. Dieses Konzept ist aufgrund der sehr kurzen Zugriffszeiten des TMS 320 zu aufwendig.

Der Verfasser realisierte deshalb eine Hardware mit umschaltbarem Speicher (*Bild 3*). Dieser Speicher besteht aus zwei 4-K×16-Bit-Blöcken, die umschaltbar jeweils einem der Rechner zugeordnet sind. Die Verwaltung der Zuordnung übernimmt der Host-µP. Im einfachsten Fall werden Seitenzuordnung des Umschaltspeichers und Reset-Anschluß des TMS 320 gesteuert.

Der Umschaltspeicher stellt sich dem Host-µP als RAM mit 8 KByte dar. Er wird über die höherwertigen Adreßleitungen selektiert. Bei einem vollständig mit RAMs bestückten Host-µP ist eine Bankadressierung oder Deselektierung erforderlich. Im realisierten Fall ist der TMS 320 an ein System mit dem 16-Bit-µP TMS 9900 gekoppelt.

Die Kopplung an andere 16-Bit-Prozessoren ist problemlos, da jeder µP die Möglichkeit zum Anschluß externer Speicher besitzt. Zum Anschluß an 8-Bit-Systeme wird die niederwertige Adreßleitung zur Selektion des jeweiligen Bytes verwendet.

1.3 Betrieb des Coprozessors

Ein Beispiel zeigt die Arbeitsweise des Coprozessors: Als Aufgabenstellung wird ein Programm für den TMS 320 zugrunde gelegt, das vor dem Start alle Eingangswerte benötigt und nach Beendigung die Ausgangswerte übergibt. Der Ablauf für den TMS 320 ist in *Bild 4* beschrieben.

Für eine zu wiederholende Abarbeitung des Programms für den TMS 320 lädt der Host-µP einmalig den Programmcode in beide Seiten des Umschaltspeichers. Nach Laden der Eingangswerte und Verarbeitung der Ausgangswerte (im Wiederholungsfall) schaltet der Host-µP die Seitenzuordnung um und setzt den Reset des TMS 320 auf „1" (Programmstart). Mit der dem Host-µP jetzt zugewandten Seite des Umschaltspeichers wird entsprechend verfahren.

Nach jedem Wechsel der Seitenzuordnung stehen also die Ergebnisse des letzten Programmlaufes zur Verfügung, die aktuellen Daten werden gerade auf der zweiten Seite vom TMS 320 verarbeitet und die Eingangswerte für den nächsten Programmlauf können geladen werden.

Der Vorteil des Umschaltspeichers, der wie eine Art Pipeline wirkt, liegt in dem ungehinderten und gleichzeitigen Arbeiten von Coprozessor (oder Coprozessoren) und dem Host-µP.

2 FFT-Algorithmen mit dem TMS 320

2.1 Grundlagen der FFT

Die FFT ist ein redundanzverminderter Algorithmus der DFT (Diskrete Fourier Transformation). Für die komplexe Eingangswertefolge der Länge N lautet die DFT:

$$F(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{-nk}$$

mit $k,n = 0...(N-1)$; $W_N^{-nk} = e^{-j \cdot \frac{2\pi n k}{N}}$

Unter der Vorraussetzung, daß N keine Primzahl ist, läßt sich die DFT in Teiltransformationen aufteilen. Für den Fall, daß N eine Potenz von 2 ist, läßt sich durch eine fortgesetzte Aufspaltung für das Beispiel N=8 die DFT durch den in *Bild 5* gezeigten Signalflußgraphen darstellen.

Ein Verfahren für die Berechnung der DFT nach Bild 5 ist ein „decimation in time, radix 2"-FFT-Algorithmus, der für die DFT, statt ursprünglich N^2, nunmehr lediglich $N \cdot \log_2 N$ komplexe Multiplikationen benötigt. Die Elementaroperation der „radix 2" FFT besteht in der Ausführung einer sogenannten „Butterfly" (s. Bild 5).

Unter Ausnutzung der Symmetrie $W_N^{-w + \frac{N}{2}} = -W_N^{-W}$ läßt sich die in *Bild 6* gezeigte Vereinfachung schreiben. Zur Ausführung der Butterflyoperation nach Bild 6 sind eine komplexe Multiplikation und zwei komplexe Additionen erforderlich, die innerhalb der FFT $N/2 \cdot \log_2 N$ mal auszuführen sind.

Die Ordnung von Eingangs- zu Ausgangswerten ergibt sich bitrevers (der binär dargestellte Laufparameter muß von hinten gelesen werden). Für die folgerichtige Anordnung müssen die Eingangswerte umgeordnet werden. Es lassen sich Signalflußgraphen ohne diesen Nachteil angeben. Die bitreverse Anordnung ist verbunden mit dem Vorteil der „In-Place"-Verarbeitung der Butterflyoperationen. Dabei lassen sich die Eingangswerte von den Ausgangswerten überschreiben. Man benötigt den Speicher für die Transformationswerte nur einmal.

2.2 *Implementierung auf dem TMS320*

Die oben beschriebene Butterflyoperation ist der Kernteil der FFT-Algorithmen. Diese Operation wird innerhalb der Transformation mehrfach ausgeführt. Der Programmcode für die Butterflyoperation nach *Bild 7* benötigt als Initialisierung die aktuellen Wurzelwerte in WR, WI sowie die Adressen ADUP, ADDW des ersten und zweiten Realteils der Eingangswerte. Die zugehörigen Imaginärteile stehen in Adresse+1. Das Programm für die Transformation besteht aus einem Rahmenprogramm (*Bild 8*) und dem Unterprogramm für die Butterflyoperation. Das Rahmenprogramm errechnet die Adressen der jeweiligen Eingangswerte und ruft nach dem Laden der zugehörigen Wurzelwerte das Butterfly-Unterprogramm auf.

2.3 *Ausführungszeit und Programmcodelänge*

Normalerweise mißt man den Aufwand für die Transformationsberechnung anhand der Anzahl der notwendigen Multiplikationen. Dieser Ansatz ist aufgrund der extrem kurzen Ausführungszeit für die Multiplikation nicht mehr zulässig. Die zur Transformation ohnehin noch erforderlichen Additionen und Adreßberechnungen tragen in erheblichem Maße zur Ausführungszeit bei.

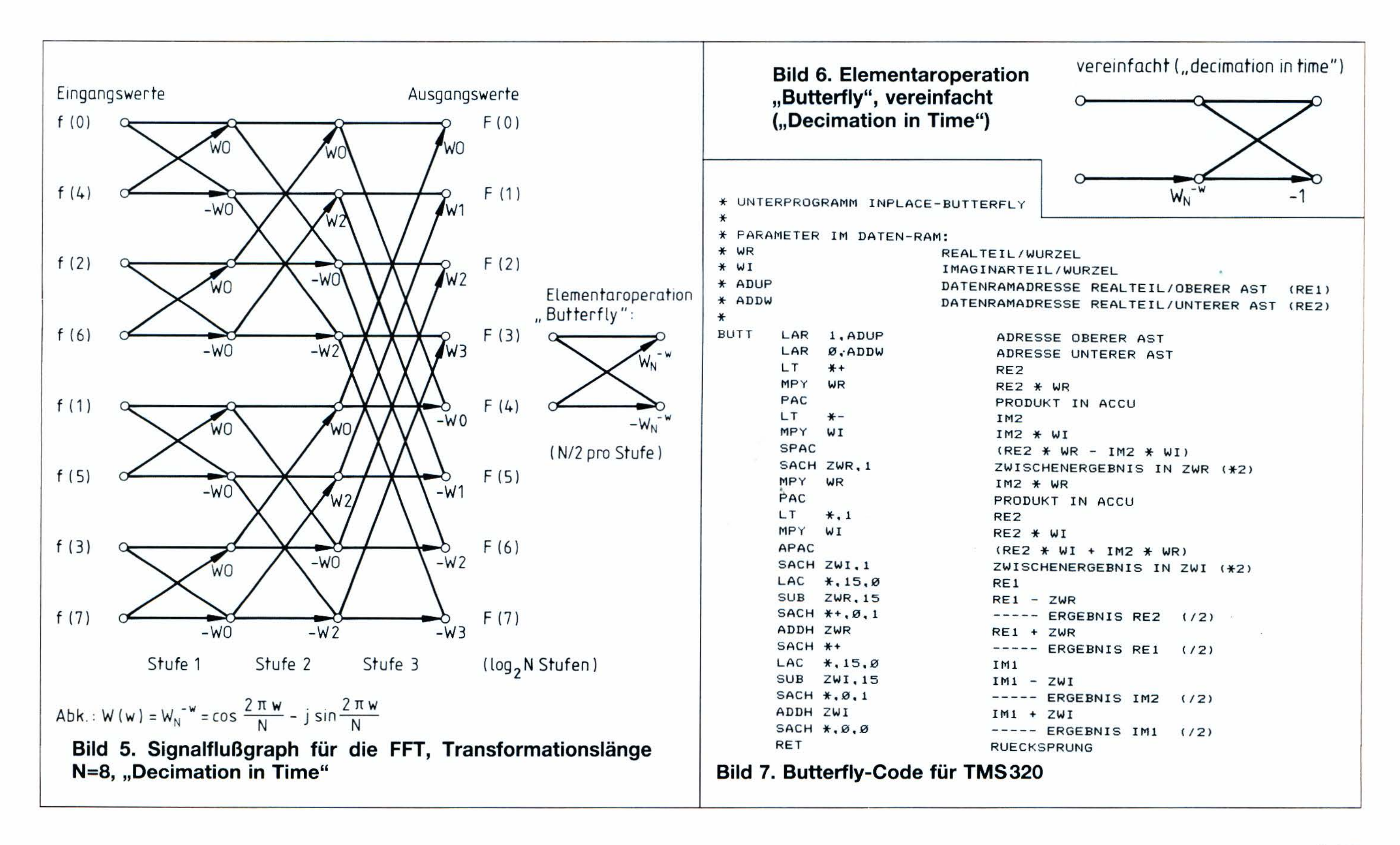


Bild 5. Signalflußgraph für die FFT, Transformationslänge N=8, „Decimation in Time"

Bild 6. Elementaroperation „Butterfly", vereinfacht („Decimation in Time")

```
* UNTERPROGRAMM INPLACE-BUTTERFLY
*
* PARAMETER IM DATEN-RAM:
* WR                         REALTEIL/WURZEL
* WI                         IMAGINARTEIL/WURZEL
* ADUP                       DATENRAMADRESSE REALTEIL/OBERER AST  (RE1)
* ADDW                       DATENRAMADRESSE REALTEIL/UNTERER AST (RE2)
*
BUTT   LAR   1,ADUP          ADRESSE OBERER AST
       LAR   Ø,ADDW          ADRESSE UNTERER AST
       LT    *+              RE2
       MPY   WR              RE2 * WR
       PAC                   PRODUKT IN ACCU
       LT    *-              IM2
       MPY   WI              IM2 * WI
       SPAC                  (RE2 * WR - IM2 * WI)
       SACH  ZWR,1           ZWISCHENERGEBNIS IN ZWR (*2)
       MPY   WR              IM2 * WR
       PAC                   PRODUKT IN ACCU
       LT    *,1             RE2
       MPY   WI              RE2 * WI
       APAC                  (RE2 * WI + IM2 * WR)
       SACH  ZWI,1           ZWISCHENERGEBNIS IN ZWI (*2)
       LAC   *,15,Ø          RE1
       SUB   ZWR,15          RE1 - ZWR
       SACH  *+,Ø,1          ----- ERGEBNIS RE2  (/2)
       ADDH  ZWR             RE1 + ZWR
       SACH  *+              ----- ERGEBNIS RE1  (/2)
       LAC   *,15,Ø          IM1
       SUB   ZWI,15          IM1 - ZWI
       SACH  *,Ø,1           ----- ERGEBNIS IM2  (/2)
       ADDH  ZWI             IM1 + ZWI
       SACH  *,Ø,Ø           ----- ERGEBNIS IM1  (/2)
       RET                   RUECKSPRUNG
```

Bild 7. Butterfly-Code für TMS320

Tabelle der FFT-Ausführungszeiten mit dem TMS320 (20-MHz-Takt)

N	Ausführungszeit einschl. TBLR/TBLW u. bitrev. Ordnen (µs)	reine Transformationszeit (µs)	Programmcodelänge einschl. Tabelle (Wörter, 16 Bit)	Art der Programmierung
8	63,6	30,4	245	direkt codiert
16	158,0	92,8	645	direkt codiert
32	383,6	254,4	1629	direkt codiert
64	975,0	754,8	3844	direkt codiert
64	1897	1575	243	Unterprogramm

N: Transformationslänge (komplex)

Die Programmierung mit einer Subroutine für die Butterflyoperation läßt sich mit getrennten Unterprogrammen für die Sonderfälle der Wurzelwerte (WI=0, WR=0) optimieren. In diesen beiden Fällen wird der Butterflycode erheblich kürzer.

Die schnellsten Algorithmen erzeugt man jedoch durch Codierungen ohne Unterprogramme. In diesen Fällen werden die Butterflycodes in direkter Adressierung so oft hintereinander geschrieben wie es die Transformation verlangt. Diese Programmierung erzeugt allerdings einen entsprechend langen Programmcode (*Tabelle*). Für Transformationslängen über N=64 wird ein mehrfacher Zwischentransfer von Teilergebnissen in den Programmspeicher des TMS320 notwendig, da der Datenspeicher des TMS320 lediglich 144 Worte groß ist. Diese Algorithmen werden sinnvollerweise auf kleinere Transformationslängen zurückgeführt und können mit mehreren TMS320 ausgeführt werden.

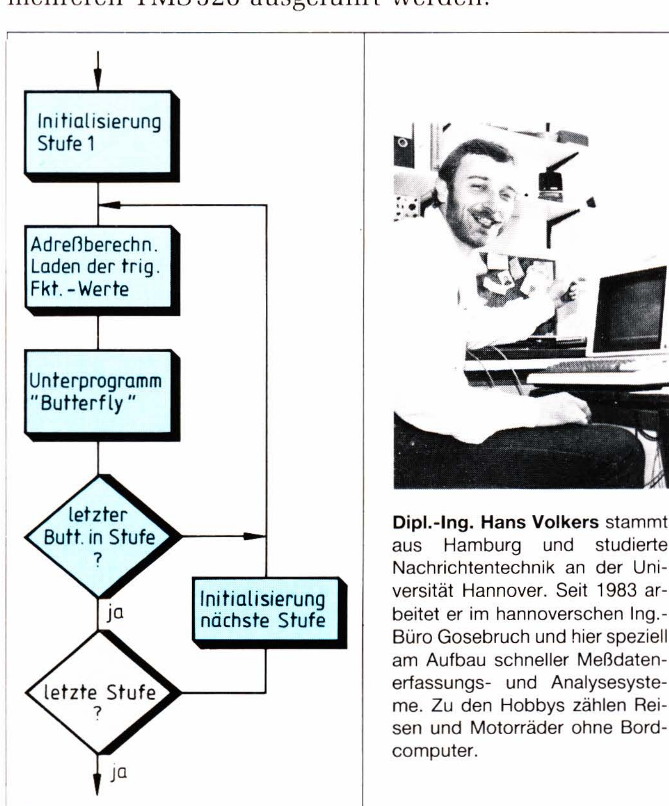


Bild 8. FFT-Rahmenprogramm

Dipl.-Ing. Hans Volkers stammt aus Hamburg und studierte Nachrichtentechnik an der Universität Hannover. Seit 1983 arbeitet er im hannoverschen Ing.-Büro Gosebruch und hier speziell am Aufbau schneller Meßdatenerfassungs- und Analysesysteme. Zu den Hobbys zählen Reisen und Motorräder ohne Bordcomputer.

2.4 Multiprozessing mit mehreren TMS320

Eine Übersichtsskizze des Hardwareaufbaus zeigt *Bild 1*. Das eigentliche Problem besteht in der Aufteilung der Gesamtaufgabe in mehrere unabhängige Teilaufgaben. Hierfür stellen FFT-Algorithmen ein gutes Beispiel dar. So läßt sich beispielsweise eine 1024-Punkte-FFT durch zweimalige Abarbeitung von jeweils 32 Transformationen der Länge N=32 ausführen. Hierbei können die ersten sowie die zweiten 32 Transformationen gleichzeitig, also auf mehreren TMS320, ausgeführt werden. Nach den ersten Transformationen muß der Host-µP Teilergebnisse zwischen den Coprozessoren umladen. Hierfür brauchen die TMS320 aufgrund des Umschaltspeichers nicht angehalten zu werden, sondern können auf ihrer Speicherseite neue Eingangsdaten bearbeiten.

Für das Beispiel einer komplexen 1024-Punkte-FFT und einem System mit vier Prozessoren TMS320 ergeben sich folgende Ausführungszeiten:

– Zeit für die ersten Teiltransformationen: 2,07 ms
– Zeit für die zweiten Teiltransformationen: 3,37 ms
– Zeit für gesamten TBLR/TBLW-Datentransfer: 1,66 ms
insgesamt: 7,10 ms

Die Frage, wie viele TMS320 parallel arbeiten können, hängt von der Aufteilung der Transformation ab. Allerdings liegt man bereits bei vier TMS320 für das Problem der Fouriertransformation in der Nähe von leistungsfähigen Arrayprozessoren. Deshalb wird für die Frage nach der noch sinnvollen Anzahl von Coprozessoren neben wirtschaftlichen Gesichtspunkten auch die Leistungsfähigkeit des Host-µP entscheidend sein.

3 Erfahrungen und Einschätzungen

Das Konzept entstand im Rahmen des DFG-Projektes „Netzrückwirkungen" am Institut für Elektrische Energieversorgung der Universität Hannover. Der TMS320 ist in beschriebener Weise an ein TMS9900-System gekoppelt. Der Coprozessor wird für eine mehrkanalige „Online"-Spektralanalyse eingesetzt und arbeitet seit etwa einem Jahr einwandfrei.

Eine wesentliche Eigenschaft des beschriebenen Konzeptes ist die einfache und übersichtliche Art der Rechnerkopplung. Diese Eigenschaft wird gerade bei der Programmierung, und hier vor allen Dingen bei Multiprozessorsystemen, wichtig.

Literatur

[1] *Oppenheim, A. V., Schafer, R. W.:* Digital Signal Processing. Prentice Hall, 1975.
[2] Firmenschriften von Texas Instruments: TMS 32010 Digital Signal Processor; Functional Specifications, May 1983. TMS 32010 Assembly Language Programmers Guide, 1983. TMS 32010 Users Guide, 1983.
[3] *Kolb, H. J.:* Prozessorkonzept zur digitalen Signalverarbeitung. ELEKTRONIK 1982, H. 21, S. 107...114.
[4] *v. Bechen, P.:* 32-Bit-µC für Signalverarbeitung und Prozeßsteuerung. ELEKTRONIK 1982, H. 22, S. 139...141.
[5] *Loges, W.:* Schneller digitaler Regler mit Signalprozessor. ELEKTRONIK 1983, H. 19, S. 51...54.
[6] *Mehrgardt, S.:* Universelles Prozessorkonzept zur digitalen Signalverarbeitung. ELEKTRONIK 1984, H. 3, S. 49...53.

Dr. Sönke Mehrgardt

Universelles Prozessorsystem zur digitalen Signalverarbeitung

Dieser Beitrag stellt ein Prozessorsystem vor, das für den universellen Einsatz in der Signalverarbeitung entwickelt wurde. Es besteht aus maximal vier eigenständigen Signalprozessoren des Typs TMS 320 (Texas Instruments), die über einen großen gemeinsamen Speicher (2 M Wörter zu je 16 Bit) gekoppelt sind. Das System wird bereits für sehr unterschiedliche Aufgaben eingesetzt. Dazu gehören die Realisierung von Array-Prozessoren und digitalen Filtern sowie die Sprachverarbeitung in Echtzeit.

1 Überlegungen beim Aufbau größerer Systeme

Der TMS 320 ist als Einchipprozessor für kleinere Systeme besonders geeignet. Wegen der hohen Prozessorleistung und wegen des hohen erreichbaren Datendurchsatzes ist dieser Baustein jedoch gerade auch für größere Systeme attraktiv. Dabei ergeben sich aber einige Schwierigkeiten aus der hohen Prozessorgeschwindigkeit (und der daraus folgenden geringen Zugriffszeit) sowie der speziellen Architektur.

Bild 1 zeigt den Zeitablauf eines Eingabevorganges (Befehl „IN"). Wie zu ersehen ist, stehen zwischen der Ausgabe der Adresse (Portadresse) und dem geforderten Anliegen der Daten am Datenbus nur 100 ns zur Verfügung. Geht man weiter davon aus, daß die Adressen sowohl auf einer Prozessorplatine als auch bei dem angesprochenen Gerät durch je ein TTL-Gatter verzögert werden und daß auch die Daten rückwärts wieder durch zwei Gatter laufen müssen, so bleiben, grob geschätzt, nur noch etwa 60 ns an Zugriffszeit übrig. In dieser Zeit muß die Adresse decodiert und der entsprechende Baustein aktiviert werden. Ein direkter Zugriff auf integrierte Bausteine der üblichen Mikroprozessor-Baureihen scheidet damit aus, und es ist in diesem Fall eine geeignete Zwischenspeicherung der Daten vorzunehmen.

Ein weiteres wesentliches Problem entsteht bei Verwendung des TMS 320, wenn eine größere Anzahl von Ein-/Ausgabegeräten angeschlossen wird (die Architektur erlaubt je acht Ein- und Ausgabeadressen). Verlangen nämlich diese Geräte zu unterschiedlichen Zeiten die Aufmerksamkeit des Prozessors, so ist nur schwer festzustellen, welches als nächstes zu bedienen ist. Für einfache Fälle stehen beim TMS 320 der Interrupteingang $\overline{\text{INT}}$ sowie ein testbarer Eingang $\overline{\text{BIO}}$ (testbar mit dem bedingten Sprung BIOZ, „Branch on IO-Status Zero") zur Verfügung. Bei mehr als zwei Geräten sind jedoch sehr zeitraubende Abfrageverfahren anzuwenden: Eine Abfrage benötigt 2...3 µs – untragbar in schnellen Echtzeitanwendungen!

Zur Lösung dieser Probleme wurde der Befehlssatz des TMS 320 durch eine einfache Zusatzschaltung „ergänzt". Sie ermöglicht es, mit nur einem BIOZ-Befehl (von 400 ns Dauer) gezielt eine von 128 Statusleitungen abzufragen. Gleichzeitig wird noch das logische Komplement dieses Befehles (also „BIOH": Sprung, falls die Statusleitung auf logisch Eins liegt) verfügbar.

Die Erweiterung ist möglich, weil der Befehlscode des Befehls BIOZ acht ungenutzte Bits enthält. *Bild 2* zeigt den Code dieses Befehles: Der Befehl besteht aus zwei 16-Bit-Worten. Das erste Wort enthält in den oberen acht

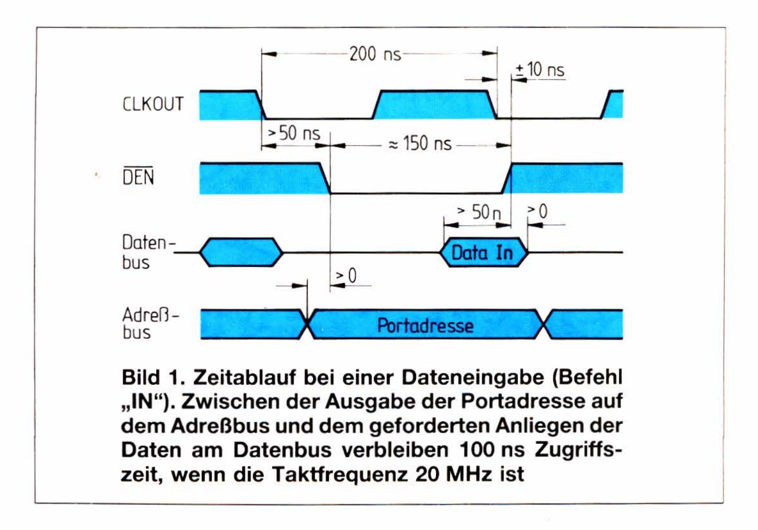


Bild 1. Zeitablauf bei einer Dateneingabe (Befehl „IN"). Zwischen der Ausgabe der Portadresse auf dem Adreßbus und dem geforderten Anliegen der Daten am Datenbus verbleiben 100 ns Zugriffszeit, wenn die Taktfrequenz 20 MHz ist

Steckbrief des Signalprozessors TMS 320

- Befehlszykluszeit 200 ns
- Wortlänge für Daten und Befehle 16 Bit
- Interner Programmspeicher 1,5 K Wörter
- Externer Programmspeicher bis zu 4 K Wörter
- 16×16-Bit-Multiplizierer (200 ns)
- 32-Bit-Rechenwerk und Akkumulator
- Parallele 16-Bit-Ein-/Ausgabe mit bis zu 2,5 M Wörtern/s

Bits den eigentlichen Befehlscode, während im zweiten Wort die Sprungadresse steht. Die freien acht Bits des ersten Befehlswortes werden nun wie folgt definiert: Sieben Bits bilden die Statusadresse und ermöglichen damit die 128 verschiedenen Statusleitungen. Das achte Bit entscheidet, ob bei Eingangssignal high oder low gesprungen werden soll.

Eine Nutzung der acht freien Bits wird bei externem Programmspeicher möglich. *Bild 3* zeigt den Zeitablauf des Befehls „BIOZ“: Synchron zum Systemtakt CLKOUT bzw. zum Signal MEN werden die beiden Worte des Befehles eingelesen. Während des Zugriffes auf das zweite Befehlswort (die Sprungadresse) testet der TMS 320 den Eingang BIO. Je nach Signal an diesem Eingang wird dann mit dem nächsten Befehl fortgefahren (BIO = 1) oder zur angegebenen Adresse gesprungen (BIO = 0). Es ist nun offenbar leicht möglich, daß jedes am Datenbus angeschlossene Gerät kontinuierlich die Daten verfolgt und immer dann die BIO-Leitung betätigt, wenn es durch die letzten, unteren Datenbits der Befehlsworte adressiert ist.

Das Prinzip wird deutlich anhand der einfachen, in *Bild 4* gezeigten Schaltung: Die vier Auswahleingänge eines Decodierers (74150) sind direkt mit dem Datenbus verbunden. Wird nun ein „BIOZ“-Befehl gelesen, so entscheiden die letzten vier Bits des Befehles, welcher der 16 Eingänge selektiert wird. Der Ausgangspegel des Decodierers wird in einem D-Flipflop synchron zum Signale MEN gespeichert, um sicherzustellen, daß das selektierte Statussignal während der gesamten Testphase (d. h. während des Zugriffes auf die Sprungadresse, siehe Bild 3) ordnungsgemäß am Eingang BIO anliegt.

Zusätzlich wurde die Schaltung durch ein Exklusiv-ODER-Gatter ergänzt, das zwischen 74150 und Dateneingang des Flipflops liegt. Abhängig vom Datenbit 7 leitet dieses Gatter den Ausgang des Decodierers invertiert oder unverändert weiter. Einige Beispiele von Befehlscodes und zugehörigen Sprungbedingungen zur Illustration:

Befehlscode (hexadezimal)	Sprungbedingung
F600	Leitung 0 low
F680	Leitung 0 high
F60F	Leitung 15 low
F68F	Leitung 15 high

Im Prozessorsystem enthalten eine Prozessorplatine sowie alle Ein- und Ausgabegeräte die nötige Logik, um die 128 adressierbaren Statusleitungen direkt testen zu können. Im Gegensatz zur Abfragemethode (2...3 µs je Abfrage) läßt sich jetzt mit einem einzigen 400-ns-Befehl überprüfen, ob ein Gerät Daten benötigt, Daten zur Verfügung hat oder eine andere Art von Bedienung erfordert. Diese Verkürzung des Zeitaufwandes auf ein Fünftel trägt ganz wesentlich zur Zeitersparnis beim Ablauf größerer Programme bei.

2 Die Prozessorplatine

Das gesamte Prozessorsystem besteht aus einigen Europakarten, die über einen Systembus verbunden sind. Bis zu vier Prozessoren befinden sich jeweils auf einer gesonderten Prozessorplatine. Neben dem Prozessor selbst enthält die Platine

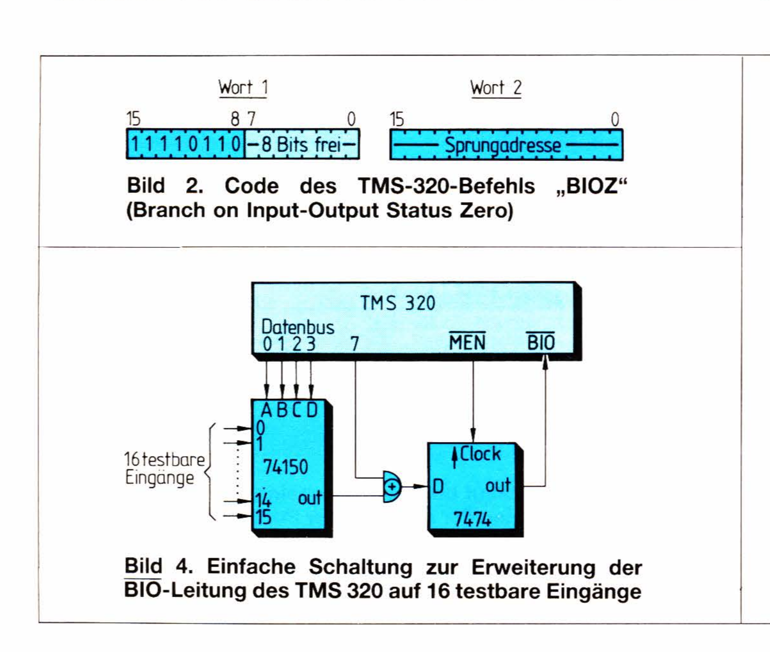


Bild 2. Code des TMS-320-Befehls „BIOZ“ (Branch on Input-Output Status Zero)

Bild 4. Einfache Schaltung zur Erweiterung der BIO-Leitung des TMS 320 auf 16 testbare Eingänge

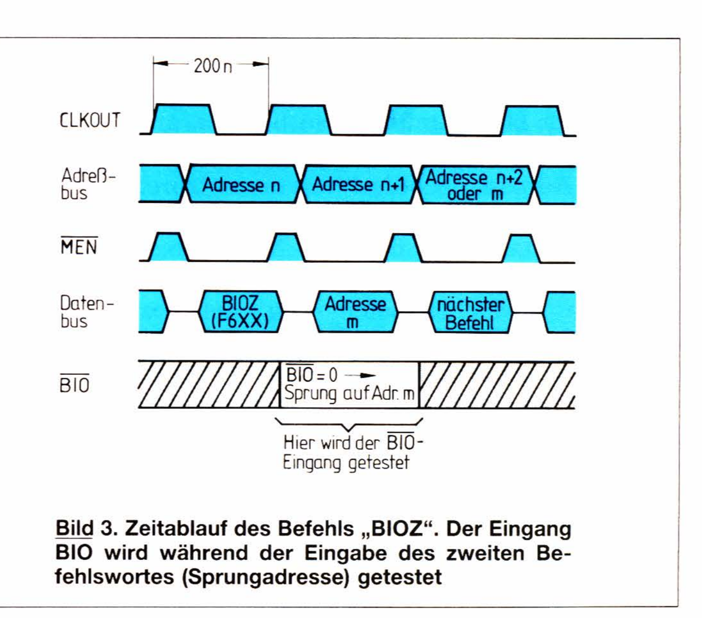


Bild 3. Zeitablauf des Befehls „BIOZ“. Der Eingang BIO wird während der Eingabe des zweiten Befehlswortes (Sprungadresse) getestet

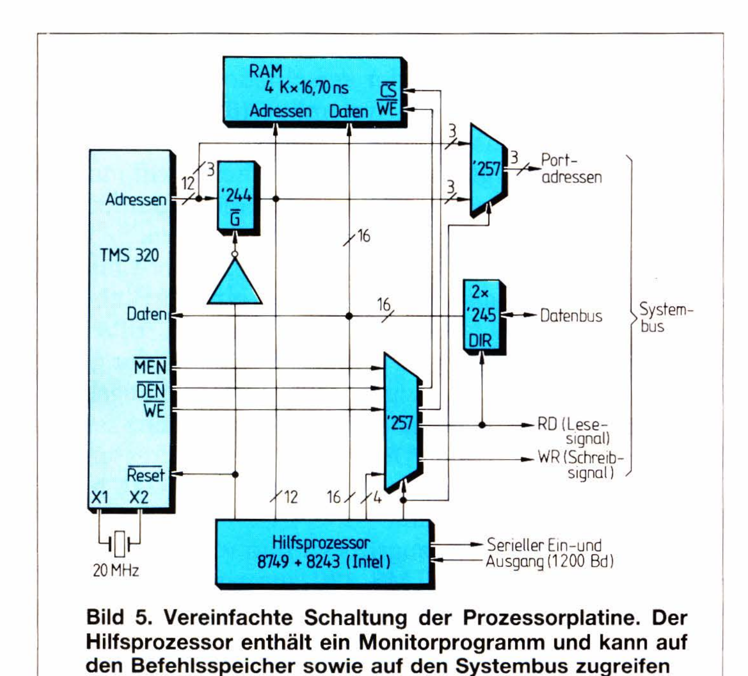


Bild 5. Vereinfachte Schaltung der Prozessorplatine. Der Hilfsprozessor enthält ein Monitorprogramm und kann auf den Befehlsspeicher sowie auf den Systembus zugreifen

- einen 4-K- × 16-Bit-Programmspeicher mit 70 ns Zugriffszeit,
- die nötige Steuerlogik zur Kommunikation mit dem Systembus,
- einen Hilfsprozessor (Intel 8749) mit einem Monitorprogramm.

Die Blockschaltung der Platine zeigt *Bild 5*. Im „normalen" Betrieb greift der TMS 320 über den Datenbus auf die Befehle im Speicher zu. Wegen der hohen Geschwindigkeit sind dabei Speicherbausteine mit etwa 70 ns Zugriffszeit nötig. Bei Eingabe- bzw. Ausgabebefehlen dient dieser Bus außerdem zum Transfer der Daten. Entsprechend der Übertragungsrichtung werden dabei die Treiberbausteine umgeschaltet, die den internen Datenbus mit dem Systembus verbinden.

Der zusätzliche Hilfsprozessor 8749 hat unterschiedliche Aufgaben. Wird die Platine als Einplatinencomputer (und auch zur Programmentwicklung) eingesetzt, so enthält der Hilfsprozessor ein Monitorprogramm: Befehle an den Monitor können über ein Terminal eingegeben werden, das an die serielle Datenleitung (1200 Bd) angeschlossen ist. Diese Befehle interpretiert das Monitorprogramm und führt sie aus. Eventuelle Ausgaben werden zum Terminal zurückübertragen.

Der Hilfsprozessor kann sowohl auf den Programmspeicher des TMS 320 als auch auf den Systembus zugreifen (er muß dazu allerdings kurzzeitig den TMS 320 mit der Reset-Leitung anhalten). Im Monitorprogramm sind folgende Befehle enthalten:
- Darstellen bzw. Ändern des Programmspeichers, der internen Register des TMS 320 und der Ports,
- Anhalten des TMS 320 bzw. Starten eines Programms,
- Setzen eines Unterbrechungspunktes zur Fehlersuche,
- Transfer eines Programmes in den Programmspeicher („Booten").

Besonderer Wert bei der Ausgestaltung des Monitorprogrammes wurde darauf gelegt, daß neben dem üblichen Eingabe- und Ausgabeformat (hexadezimal, oktal) auch die Zahlendarstellungen möglich sind, in denen sich bei Anwendungen der digitalen Signalverarbeitung „leichter denken" läßt. Entsprechend sind alle Ein- und Ausgaben auch dezimal mit Rechtskomma- (ganzzahlig) bzw. der Linkskommadarstellung (als Dezimalbruch) erlaubt.

Wird die Prozessorplatine stets mit dem gleichen Programm verwendet, so kann der Hilfsprozessor statt des Monitors die nötigen Daten enthalten, die er nach Einschalten des Gerätes in den Programmspeicher lädt. Danach startet er den TMS 320 an geeigneter Stelle, so daß die Platine sofort einsatzbereit ist. In dieser Variante betreiben wir die Prozessorplatine in unserem System: Der Programmspeicher wird nach Einschalten mit einem kleinen Programm (dem „Urlader") initiiert, mit dem der TMS 320 dann die restlichen Befehle über den Systembus in seinen Programmspeicher lädt.

Die gesamte Hardware mit beiden Prozessoren, dem Programmspeicher, der Steuerlogik und den Treiberbausteinen für den Systembus bzw. die serielle Datenleitung zum Terminal paßt auf eine Europakarte. Die aufgebaute Schaltung zeigt *Bild 6*.

3 Ein- und Ausgabegeräte

Das System wird primär zur Verarbeitung analoger Daten eingesetzt. Digitale Daten sind nur im Ausnahmefall (zum Beispiel beim Einsatz als Array-Prozessor an einem Großrechner) verfügbar. Zur digitalen Verarbeitung analoger Daten stehen deshalb 16-Bit-Digital-/Analog-Umsetzer und -Analog-/Digital-Umsetzer zur Verfügung. Sie sind, genau wie die Prozessorplatine, auf jeweils einer Europakarte verwirklicht.

Beide Umsetzertypen sind mit FIFO-Speichern (16 × 16 Bit) ausgerüstet. Diese Zwischenspeicher ermöglichen es dem Programmierer, gelegentlich auch längere Zwischenrechnungen durchzuführen, ohne daß die streng zeitäquidistante Erzeugung und Abfrage von Abtastwerten unterbrochen wird.

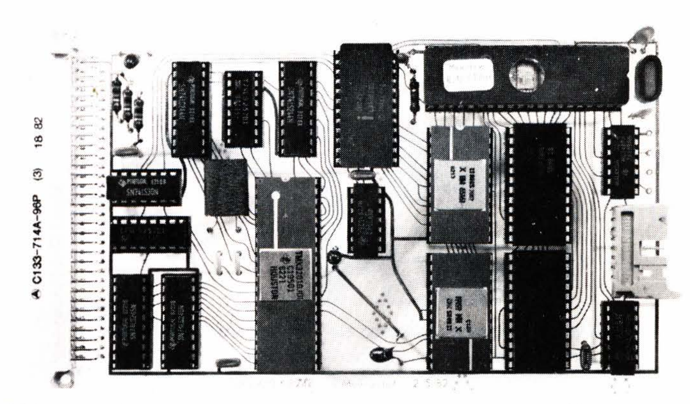


Bild 6. Die Prozessorplatine ist in Europaformat ausgeführt

Zur Kopplung des Prozessorsystems mit anderen Geräten ist ferner ein paralleles Interface (16 Bit) vorhanden. Je nach Anwendung kann es konfiguriert werden für eine bidirektonale 16-Bit-Leitung oder für getrennte 16-Bit-Ausgabe- und -Eingabeleitungen. Eine entsprechende Steuerlogik synchronisiert dabei den Datenaustausch mit dem externen Gerät. Die Übertragungsgeschwindigkeit beträgt bis über 1 MWörter/s. Über dieses Interface ist das Prozessorsystem mit einem Großrechner verbunden, dem das System als intelligentes Ein- und Ausgabegerät dient und von dem es in speziellen Fällen als Array-Prozessor eingesetzt wird.

4 Externer Speicher und Prozessorkopplung

Der TMS 320 verfügt über einen internen Datenspeicher von 144 Wörtern (zu je 16 Bit). Auf diesen Speicher kann mit den in 200 ns abgearbeiteten Befehlen des Bausteines direkt zugegriffen werden. Reicht er für eine Anwendung nicht mehr aus, so ist mit zwei speziellen Befehlen (TBLR, TBLW) ein externer Speicher von maximal 4 K × 16 Bit ansprechbar. In vielen Fällen ist jedoch auch dieser Bereich noch zu klein, und es muß eine andere Lösung gewählt werden.

Bei dem beschriebenen Prozessorsystem wurde eine Konfiguration gewählt, in der auf einen großen Speicherbereich (4 MByte) mehrere Prozessoren im Zeitmultiplex zugreifen können. Neben dem sehr großen Speicherbereich hat man dabei gleichzeitig den Vorteil, durch Kopplung einiger Prozessoren die Verarbeitungsleistung des Systems vervielfachen zu können. Eine engere Kopplung der Prozessoren ist übrigens nur schwer realisierbar, da der TMS 320 über keine entsprechenden Hardwareeigenschaften verfügt.

Das Prinzip der Prozessorkopplung zeigt *Bild 7*: Die einzelnen Prozessoren sind mit einem getrennten Speicherbus über Controller verbunden. Diese Controller sorgen für die Anpassung der verschiedenen Zugriffsgeschwindigkeiten auf dem Systembus (60 ns) und dem Speicherbus (300 ns), und sie müssen im Falle eines Zugriffskonfliktes (gleichzeitiger Speicherzugriff mehrerer Prozessoren) durch Prioritätsentscheidungen für die richtige Abarbeitung des Datentransfers sorgen.

Bild 7. Blockschaltung des Prozessorsystems: Die einzelnen Prozessoren sind über Speicher-Controller mit einem bis zu 4 MByte großen Speicher verbunden

Bild 8 zeigt die Blockschaltung der Speicher-Controller. Die Speicherplatinen werden über eine 21 Bit breite Adresse selektiert. Die 16 niederwertigen Bits der Adresse werden von einem bidirektionalen Zähler geliefert. Dieser Zähler kann vom TMS 320 auf vorgegebene Adressen gesetzt werden, auch ist es möglich, seinen Inhalt wieder zu lesen. Die oberen fünf Adressenbits speichert ein Steuerregister, in das der TMS 320 ebenfalls direkt schreiben kann.

Der Ablauf eines Lesezugriffs ist nun wie folgt: Zuerst wird die Adresse vom TMS 320 ausgegeben. Dadurch wird ein Speicherzugriff initiiert, indem der Controller versucht, so schnell wie möglich die gewünschten Daten in seinem Datenspeicher abzulegen. Dort stehen sie zur Verfügung, um mit einem Eingabebefehl des TMS 320 in den Prozessor eingelesen zu werden. Die Zugriffszeit des Speichers beträgt zur Zeit (bei Verwendung dynamischer RAMs) 300 ns. Sie kann bei gleichzeitigem Zugriff mehrerer Prozessoren oder durch nötige Auffrischzyklen entsprechend steigen. Der Speicherbus und die Controller sind jedoch so ausgelegt, daß bei Mischbestückung mit Speichern unterschiedlicher Geschwindigkeit stets mit dem jeweils maximalen Tempo gearbeitet werden kann (dies wird, wie üblich, durch eine von den Speicherplatinen bediente „Wait"-Leitung erreicht). Die minimal erreichbare Zugriffszeit beträgt dann 100 ns, sie ist durch verschiedene Verzögerungen der gesamten Steuerlogik bedingt.

Bei einem Schreibzugriff werden die Ausgabedaten des TMS 320 erst im Controller zwischengespeichert und dann, so bald wie möglich, über den Speicherbus zur entsprechenden Einheit transferiert.

Die sequentielle Abarbeitung größerer Datenbereiche wird vom Controller zusätzlich unterstützt: Es ist möglich, bei Schreib- und/oder Lesezugriffen die niederwertigen 16 Bit der Adresse automatisch zu inkrementieren oder zu dekrementieren. Zwischen aufeinanderfolgenden Speicherzugriffen braucht dann nicht jeweils erst die neue Adresse berechnet und ausgegeben zu werden, sondern es reicht dazu ein einziger „IN"- bzw. „OUT"-Befehl von 400 ns Dauer aus. Entsprechend läßt sich ein Speicherbereich mit der vollen Geschwindigkeit des TMS 320 von 2,5 M Wörtern/s einlesen bzw. beschreiben.

Zur weiteren Beschleunigung der Signalverarbeitung enthält der Controller eine Logik, mit der das Erreichen eines Speicherbereichendes leicht abzufragen ist. Dazu kann mit dem „BIOZ"-Befehl getestet werden, ob die im 16-Bit-Zähler gerade gespeicherte Zahl Vielfaches einer gewünschten Zweierpotenz ist, d. h., ob eine gerade Zahl, ein Vielfaches von Vier oder Acht usw., vorliegt.

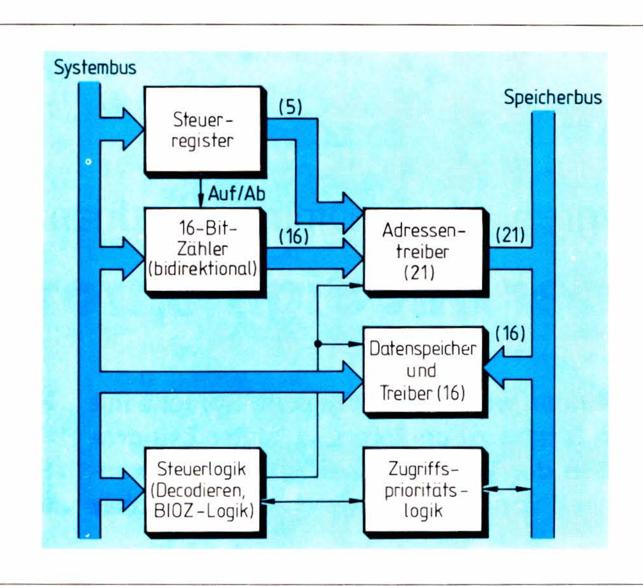


Bild 8. Blockschaltung eines Speicher-Controllers. Die 21-Bit-Adresse stellen ein bidirektionaler Zähler (16 Bit) und ein Steuerregister (5 Bit) zur Verfügung. Zur Anpassung der verschiedenen Busgeschwindigkeiten werden in beiden Übertragungsrichtungen die Daten zwischengespeichert. Eine Steuer- und eine Prioritätslogik sorgen für die Vermeidung von Buskonflikten bei simultanem Zugriff mehrerer Prozessoren

Damit läßt sich jeder 64 K × 16 Bit große Speicherbereich in beliebige Teilblöcke unterteilen, deren Grenzen ohne zusätzliche Softwareabfragen leicht erkennbar sind.

5 Das Gesamtsystem

Alle Baugruppen sind auf Europakarten realisiert, die auf einer Busplatine zur gewünschten Konstellation zusammengestellt werden können. Die maximale Ausbaustufe kann vier Prozessoren mit jeweils zahlreichen Peripheriegeräten umfassen, die auf einen gemeinsamen Speicher von 4 MByte zugreifen. Mit diesem System steht eine erhebliche Rechen- und Speicherkapazität zur Verfügung, mit der sich auch aufwendige Probleme digitaler Signalverarbeitung in Echtzeit lösen lassen.

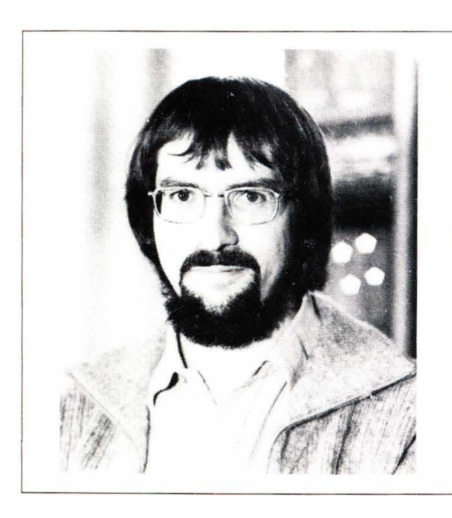
Realisiert ist zur Zeit eine etwas kleinere Konfiguration mit zwei Prozessoren. Prozessor A bedient je zwei D/A- und A/D-Umsetzer sowie ein paralleles Interface zum Anschluß an einen Großrechner. Die Verbindung zum insgesamt 256 K × 16 Bit großen Speicher stellt ein Speicher-Controller her. Diese Hälfte des Gesamtsystems arbeitet die Eingaben und Ausgaben ab und verwaltet zeitkritische Teile des jeweiligen Problems.

Prozessor B verfügt nur über die Kopplung mit dem Speicher. Er übernimmt rechenintensive Arbeiten und erhält nötige Daten über den Speicher von Prozessor A. Auf dem gleichen Weg liefert er die Rechenergebnisse zurück.

Zur Signalerzeugung für Meßzwecke reicht oft ein Prozessor schon aus. Ein Beispiel: Bei einer Abtastrate von 20 kHz kann Prozessor A acht Sinusschwingungen verschiedener Frequenz, Amplitude und Phase in Echtzeit berechnen und geeignet über die zwei D/A-Umsetzer ausgeben. Es ist damit ein zumindest für wissenschaftliche Zwecke sehr interessanter Weg der Erzeugung komplexer Klänge und Geräusche möglich.

Bei Anwendungen mit digitalen Filtern läßt sich der spezielle Befehlssatz des Prozessors besonders gut nutzen. Bei einer Abtastrate von 40 kHz (sehr gute HiFi-Qualität) lassen sich mit einem Prozessor noch Filter mit 60 Koeffizienten in Echtzeit realisieren. Damit lassen sich fast alle praktisch interessanten Übertragungsfunktionen verwirklichen.

Ebenfalls mit nur einem Prozessor lassen sich Vocoder programmieren, die ausgezeichnete Sprachqualität liefern. So war es z. B. möglich, ein für Forschungszwecke entwickeltes analoges Gerät innerhalb kurzer Zeit durch ein digitales Äquivalent (das sogar anschlußkompatibel war und die gleichen Steuersignale verarbeitete) mit dem TMS-320-System zu ersetzen.



Dr. Sönke Mehrgardt ist gebürtiger Kasseler. Er studierte Physik an der Universität Göttingen und ist dort seitdem als Assistent am dritten Physikalischen Institut tätig. Neben seinem „eigentlichen" Forschungsgebiet, der Untersuchung des menschlichen Gehörs, widmet er sich unterschiedlichen Anwendungen der digitalen Signalverarbeitung. Zu den Hobbys zählen neben den Kindern das Skilaufen und das Klavierspiel.

Literatur

[1] 32-Bit-Mikrocomputer für Signalverarbeitung und Prozeßsteuerung. ELEKTRONIK 1982, H. 22, S. 139...141.

[2] TMS 320 Preliminary Functional Specification. Texas Instruments.

Göran Nygren, M. S. Dipl.-Ing. Adrian Zoicas

Vocoder mit Signalprozessor

1. Teil

LPC-Vocoder finden wegen der Fähigkeit, Sprache mit einer geringen Bitrate zu codieren, in letzter Zeit großes Interesse. In der Vergangenheit war die Verwendungsmöglichkeit von LPC-Vocodern aufgrund hoher Kosten und umfangreicher Hardware begrenzt. Dieses Problem wurde mit Hilfe der schnellen digitalen Signalprozessoren, die seit einiger Zeit auf dem Markt sind, gelöst.

Ein weiteres Hindernis waren bisher bei der Anwendung die ziemlich komplizierten und zeitaufwendigen Algorithmen, die erforderlich sind für Echtzeit-LPC-Analyse und -LPC-Synthese. Dieses zweite Problem ist nun durch die Entwicklung eines Satzes von drei vorprogrammierten Chips gelöst, der die rechnerisch aufwendigen Teile eines LPC-Vocoder-Systems, insbesondere der LPC-Analyse, der Pitchdetektion (Stimmlageerkennung) und der LPC-Synthese ausführt. Bei den Chips handelt es sich um EPROM-Versionen des Signalprozessors µPD7720 von NEC. Die Firmware für die Signalverarbeitung wurde von Watchdog Computer Assistance AB und Linteko Elektronik AB, Täby, Schweden, entwickelt.

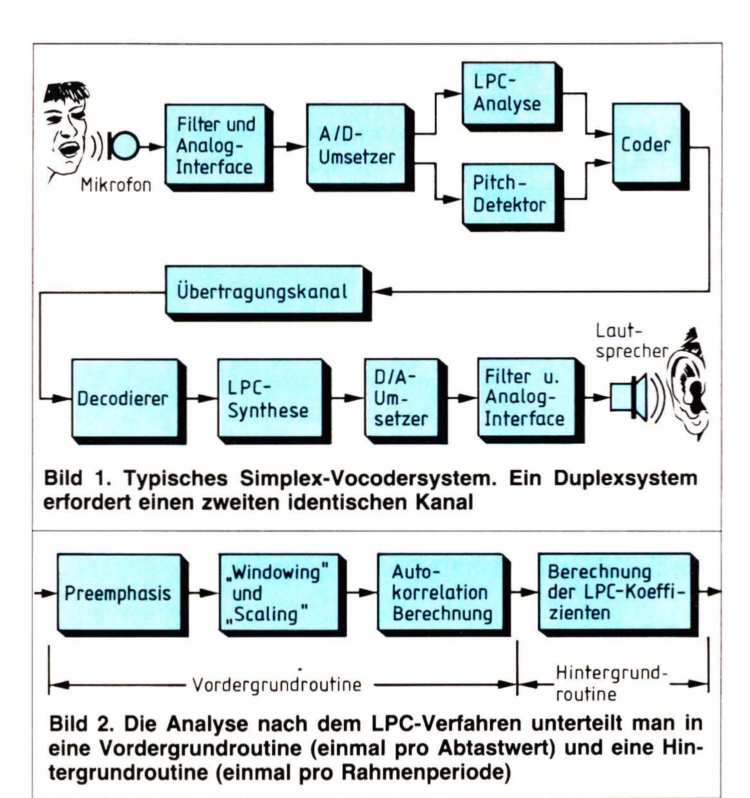


Bild 1. Typisches Simplex-Vocodersystem. Ein Duplexsystem erfordert einen zweiten identischen Kanal

Bild 2. Die Analyse nach dem LPC-Verfahren unterteilt man in eine Vordergrundroutine (einmal pro Abtastwert) und eine Hintergrundroutine (einmal pro Rahmenperiode)

1 Typisches LPC-Vocoder-System

Bild 1 zeigt die Blockschaltung eines typischen Simplex-LPC-Vocoders. Ein Duplex-System erfordert einen zusätzlichen Kanal in umgekehrter Richtung. Der Sender nimmt das Schallsignal über ein Mikrofon auf. Wenn das Mikrofon nicht in einer sehr stillen Umgebung benutzt wird, so sollte es in der Lage sein, Hintergrundgeräusche zu unterdrücken. Das Nf-Signal läuft durch eine analoge Zwischenstufe, die aus einem Vorverstärker, Verstärkersteuerung und einem Eingangstiefpaß (Anti-Aliasing-Filter) besteht.

Die A/D-Umsetzung erfolgt mit einer Abtastfrequenz von 8 kHz. Bei Linearcodierung beträgt die Auflösung 12 Bit oder 8 Bit im logarithmisch komprimierten Code (A-Law oder µ-Law). Das Signal läuft dann in den LPC-Analysator und den Pitchdetektor. Einmal pro Rahmenperiode berechnet der Vocoder eine neue Pitchschätzung (Stimmlagenschätzung) und einen neuen Satz von LPC-Koeffizienten. Diese Parameter werden quantifiziert und codiert und zusammen mit den notwendigen Synchronisationszeichen auf einem seriellen Kanal zum Empfänger geschickt. Die Übertragungsrate beträgt dabei 2,4 kBit/s.

Der Empfänger synchronisiert sich mit den empfangenen seriellen Daten. Die empfangenen Parameter werden aufgeschlüsselt sowie decodiert und vom LPC-Synthesizer benutzt, um eine Approximation des Sender-Eingangssignals zu erzeugen. Nachdem es einen D/A-Umsetzer und ein Tiefpaßfilter durchlaufen hat, wird das nun rekonstruierte analoge Signal in Schall verwandelt.

2 LPC-Analyse-Chip

Die LPC-Analyse ist unterteilt in eine Vordergrundroutine, die einmal pro Abtastperiode ausgeführt wird, und eine Hintergrundroutine, die einmal in jedem „Frame" (Rahmen) aufgerufen wird.

Die erste Routine umfaßt Preemphase, Fensterabtastung („Windowing"), Skalierung und Auto-Korrelationsrechnungen. Die Hintergrundroutine setzt sich zusammen aus einer auf Wunsch aufrufbaren Spektrumsglättungsroutine und einer LeRoux-Gueguen-Rekursion [1], um die Prädiktorkoeffizienten und die residuale Fehlerenergie zu berechnen (*Bild 2*).

Alle abgetasteten Daten, die entweder aus logarithmisch komprimierten 8-Bit-Worten oder linearen 16-Bit-Worten bestehen, gelangen zuerst in den Preemphaseblock. Die Energie in einem Sprachsignal konzentriert sich auf den Frequenzbereich unter 500 Hz (*Bild 3*). Der Sinn der Preemphase ist die Glättung des Eingangsspektrums durch Anhebung der höheren Frequenzanteile im Spektrum. Ein glatteres Spektrum erhöht die Genauigkeit der Prädiktorkoeffizienten, wenn die Berechnung mit beschränkter Wortlänge ausgeführt wird.

Der Preemphaseblock besteht aus einem einstufigen digitalen Hochpaßfilter. Nach der Multiplikation mit einer Konstanten wird der letzte Abtastwert von jedem digitalisierten Abtastwert subtrahiert. Die Konstante, die die Eckfrequenz des Filters steuert, wird vom Steuerprozessor bei der Initialisierung eingeladen.

Nach der Preemphase werden die Sprachdaten durch ein Hamming-Fenster geschickt, mit einem abwertenden Skalierungsfaktor multipliziert und der Auto-Korrelationsroutine zugeführt. Der Wert des Skalierungsfaktors hängt von dem Rahmen ab und muß so gewählt werden, daß der Korrelator nicht überläuft. Um P Prädiktorkoeffizienten zu berechnen, ist es notwendig, Zugriff zu den Auto-Korrelationskoeffizienten R (O)...R(P) zu haben. Der Analyse-Chip rechnet mit $P \leq 12$. R(n) ist definiert als Summe von R(i)*R(i-n) für alle Abtastwerte in einem Frame. Um die Genauigkeit zu erhalten, werden die Zwischensummen in Wortlängen von 32 Bit gespeichert.

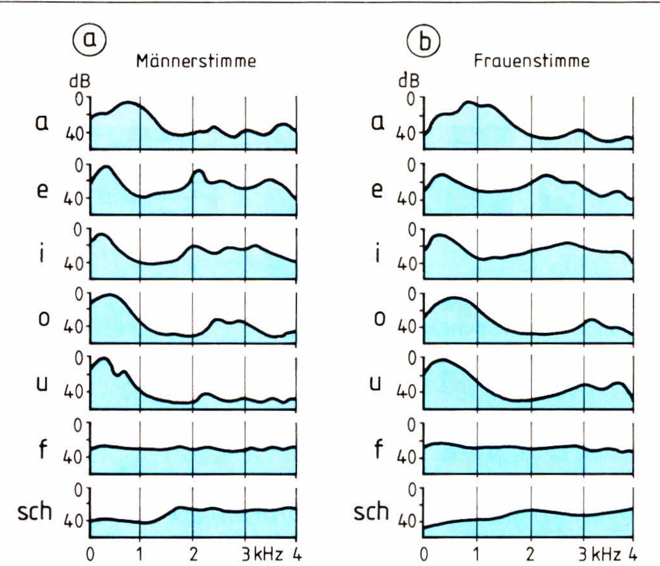


Bild 3. Spektralanalyse von Vokalen und Konsonanten, gesprochen von einer Frauen- (b) und Männerstimme (a). Die Grundfrequenz der Männerstimme beträgt ca. 140 Hz, die der Frauenstimme ca. 280 Hz. Der von den Stimmbändern erzeugte Ton ist sehr obertonreich, die unterschiedliche Amplitudenverteilung bei den verschiedenen Vokalen entsteht durch Resonanzen, sogenannte „Formanten", die beim Sprechen durch Bildung von Resonanzräumen erzeugt werden. Diese „Filterung" ist bei stimmlosen Zischlauten ebenso wichtig, die Unterschiede im „Rauschspektrum" bei den Lauten „f" und „sch" machen dies deutlich.
Bei der Untersuchung der Formantbereiche der Vokale zeigt sich, daß sie durch Resonanzen im Mund- und Rachenraum bestimmte Kolorierungen erhalten, d. h. durch die Form und Größe der Resonanzräume werden bestimmte Obertonbereiche der von den Stimmbändern erzeugten Klänge in für jeden Vokal typischer Weise betont oder gedämpft. Das ist der Grund, daß durch Veränderung des Resonanzraumes aus den jeweils gleichen Klängen der Stimmbänder die verschiedenen Vokale entstehen [9].

Am Ende eines Analyserahmens wird die Hintergrundroutine gestartet. Sie berechnet die LPC-Koeffizienten und die residuale Energie aus den Auto-Korrelationskoeffizienten, die von der Vordergrundroutine erzeugt wurden. Zuerst werden die Auto-Korrelationskoeffizienten normiert, auf 16 Bit gerundet und dann an eine Routine weitergereicht, die die LeRoux-Gueguen-Rekursion ausführt. Sie erzeugt die Prädiktor-Koeffizienten und die residuale Energie, die schon für die vorausgegangene Normierung korrigiert wurden. Dies sind die Ausgangsparameter für den Steuerprozessor. Um möglichst flexibel zu bleiben, führt der Analysechip keine Parametercodierung durch.

Geschichte des Vocoders

Im Jahre 1790 gelang es erstmals, ein Gerät zu entwickeln, das man heute als Sprachsynthesizer bezeichnen würde. Angetrieben durch Blasebalg und Luft ermöglichte es dieses Gerät, „synthetische" Sprache zu erzeugen. Es dauerte allerdings mehr als ein Jahrhundert, bis im Jahre 1900 eine ähnliche, jedoch durch Elektrizität betriebene, Konstruktion entstehen konnte. Dieses Exemplar sollte die Basis für die Entwicklungsarbeit **Homer Dudleys** sein, der im Jahre 1936 in den Bell Laboratories, USA, seinen Vocoder schuf. An dem technischen Konzept dieses „Voice Coders" änderte sich bis zur Gegenwart nichts mehr.

Natürlich läßt heute der fortgeschrittene Entwicklungsstand der gesamten Elektronik Integrationsdichten zu, die zum damaligen Zeitpunkt unbekannt waren. Das Grundkonzept des „Kanalvocoding" aber existierte schon damals.

Analoge Vocoder lassen sich mittlerweile durch digitale Signalverarbeitung ersetzen. Bei gleicher Bit-Rate (2,4 kBit/s), aber mit erheblich niedrigerem rechnerischen Aufwand, empfiehlt sich der LPC-Vocoder. Neuere Algorithmen wie „RELP" (Residual Exitation Linear Prediction [6]) erlauben bessere Sprachqualität, allerdings bei höheren Übertragungsraten (9,6 kBit/s).

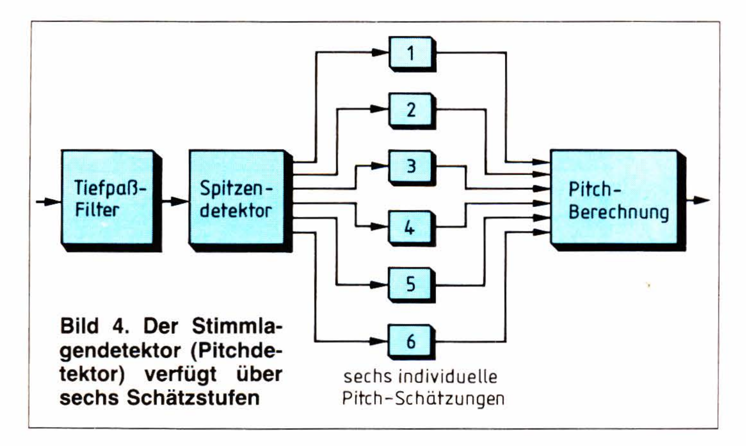


Bild 4. Der Stimmlagendetektor (Pitchdetektor) verfügt über sechs Schätzstufen



Göran Nygren stammt aus Sundsvall, Schweden. Er studierte Elektronik am Royal Institute of Technology (KTH) in Stockholm und erhielt 1979 den Master of Science. Zwischen 1979 und 1980 war er als Projekt-Ingenieur auf dem Gebiet der Medizintechnik am Korolinska-Institut tätig. Seit 1980 arbeitet er bei Watchdog Assistance, Stockholm, auf dem Gebiet der Sprachcodierung bei niedrigen Bitraten für die Sprachsynthese. Sein Hobby ist Kochen



Dipl.-Ing. Adrian Zoicas stammt aus Timisoara, Rumänien, wo er 1976 sein Elektronik-Studium am Polytechnischen Institut „Traian Vuia" absolvierte. Bei IEMI Bucuresti war er bis 1977 in der Meßtechnik tätig und anschließend beschäftigte er sich mit der Funkenerosion bei Electrotimis Timisoara. 1980 emigrierte er in die BRD und begann seine Tätigkeit bei der NEC Electronics Europe in Düsseldorf. Als Applikationsingenieur spezialisierte er sich in Sprach- und Digitalsignalverarbeitung.
Zu seinen Hobbys zählt er Sprachen, Internationales Recht, Lebensphilosophie und Tennis

Als Option kann zusätzlich eine spektrale Glättung („Spectral Smoothing Technique – SST") angewendet werden. Die SST erweitert geringfügig die Bandbreite der Formanten (siehe Kasten) dadurch, daß die Auto-Korrelationskoeffizienten vor der LeRoux-Gueguen-Rekursion durch ein Fenster geschickt werden. Bei Sprechern mit hohen Stimmlagen („High Pitched Voice"), bei denen die Unterschätzung der Formantenbandbreite oft ein Problem darstellt, ist der Einsatz der SST von besonderem Vorteil.

3 Chip für die Pitchdetektion

In jedem Analyserahmen berechnet der Pitchdetektor (*Bild 4*) eine Schätzung für die Pitchperiode. Er berechnet ebenfalls einige Hilfsparameter, um die Entscheidung, ob der Laut stimmhaft oder stimmlos ist, zu erleichtern. Für die Pitchdetektion wird ein modifizierter Gold-Algorithmus [3] verwendet. Wie bei der LPC-Analyse teilen sich die Berechnungen in eine „einmal pro Abtastung"-Vordergrundroutine und eine „einmal pro Frame"-Hintergrundroutine auf.

Zuerst werden die Abtastwerte durch ein Tiefpaßfilter geschickt. Der Steuerprozessor übergibt beim Start die Filter-Koeffizienten. Nach dem Filtern gelangen die Abtastwerte an sechs parallele Pitch-Periodenschätzer, die eine 6x6-Matrix von Pitchschätzungen erzeugen. Am Ende des Analyserahmens ist die letzte Schätzung von jedem Detektor ein Kandidat für die letzte Pitchschätzung. Jeder der sechs Werte wird mit den anderen Werten in der 6x6-Matrix verglichen. Jedesmal, wenn die Differenz unterhalb einer bestimmten Schwelle liegt, wird die Punktezahl erhöht. Der Wert bekommt einen zusätzlichen Bonus, wenn er in den erwarteten Sprachlagenbereich, basierend auf vorangegangenen Schätzungen, fällt. Der erfolgreichste „Kandidat" für die Pitchperiode und seine Punktezahl sind der Ausgang für den Steuerprozessor.

Der einzelne Faktor, der die Qualität der reproduzierten Sprache am stärksten beeinflußt, ist die Entscheidung für stimmhaften/stimmlosen Laut. Diese wichtige Entscheidung wird am besten dem Steuerprozessor überlassen, dem eine größere Menge von Entscheidungsparametern zur Verfügung stehen kann [4].

Um die Entscheidung zu erleichtern, berechnet der Pitchdetektor-Baustein ebenfalls die Anzahl der Nulldurchgänge und den ersten Reflexionskoeffizienten, basierend auf den Daten ohne Preemphase, und stellt dies dem Steuerprozessor zur Verfügung.

(Fortsetzung folgt)

Anwendungen von Vocodersystemen

Bis in den letzten Jahren fanden Vocoder hauptsächlich Anwendung im Bereich der Nachrichtentechnik, z. B. zur Verschlüsselung von Informationen im militärischen Bereich. Weil sie sehr teuer waren, blieb die Zahl der verwendeten Geräte lange Zeit begrenzt. Aufgrund der Verfügbarkeit von digitalen Einchip-Signalprozessoren wie auch wegen Erweiterung des Angebotes an Modems und digitalen Netzwerken wurden die Voraussetzungen für die Einführung von Vocodern auch in zivile Applikationen geschaffen.

Ort „A"
PBX
Telefonzentrale
digitale Sprachsignale 2400 Bits
Vocoder
Computer
Daten-Modem 9600 Bit/s
Daten 7200 Bit/s

Ort „B"
PBX
Telefonzentrale
digitale Sprachsignale 2400 Bit/s
Daten-Modem 9600 Bit/s
Vocoder
Computer
Daten 7200 Bit/s

Göran Nygren, M. S., Dipl.-Ing. Adrian Zoicas

Vocoder mit Signalprozessor

2. Teil

Der erste Teil dieses Beitrages zeigt den Aufbau eines Vocoder-Systems, das in LPC-Technik arbeitet. Mit Hilfe von Signalprozessoren können die für dieses Verfahren notwendigen Funktionen ausgeführt werden. Das Schaltungskonzept erläutert der abschließende zweite Teil.

4 LPC-Synthese-Chip

Der Synthese-Chip besteht aus einem Erregergenerator mit zwei Signalquellen, einem Lattice-Filter mit wählbarer Länge (maximale Länge = 12) und einer Deemphase-Stufe (*Bild 5*). In jedem Rahmen empfängt der Synthesizer eine Energieschätzung, eine Stimmlagen-Entscheidung (stimmhaft/stimmlos) und einen Satz von Prädiktor-Koeffizienten vom Steuerprozessor.

In einem stimmlosen Rahmen wird das Lattice-Filter von Pseudo-Zufallsrauschen erregt, während in einem stimmhaften Rahmen eine Impulsfolge mit Mittelwert Null benutzt wird (*Bild 6*). Die Periode der Impulsfolge entspricht der der geschätzten Stimmlage, und die Amplitude basiert auf einer linearen Interpolation von Energieschätzungen von vergangenen sowie gegenwärtigen Rahmen. Die Amplitude wird ebenfalls durch die Stimmlage beeinflußt. Mit einer zunehmenden Zahl von Impulsen je Rahmen muß die Energie jedes Impulses reduziert werden, um die gesamte Eingangsenergie konstant zu halten. Stimmlage-, Energie- und LPC-Koeffizienten werden in jeder Pitchperiode interpoliert und aktualisiert. Um starke Wechsel des Eingangssignals zu vermeiden, wird zwischen zwei Frames mit verschiedener Erregungsquelle oder wenn ein Rahmen ohne Laut ist, nicht interpoliert.

Durch Interpolation der LPC-Koeffizienten erreicht man einen allmählichen Übergang vom Spektrum eines Rahmens zum Spektrum des nächsten Rahmens. Wenn der Unterschied zwischen zwei Sätzen von Prädiktorkoeffizienten gering ist, arbeitet die lineare Interpolation gut. Wenn jedoch der Unterschied groß ist, dann produzieren die interpolierten Koeffizienten nicht unbedingt ein Spektrum, das den Mittelwert des Eingangsspektrums darstellt. Um die Erzeugung fehlerhafter Spektren zu vermeiden, darf keine Interpolation stattfinden, wenn abrupte Wechsel in den LPC-Koeffizienten auftreten.

Auf das Lattice-Filter folgt eine Deemphase-Stufe, die die Preemphase im LPC-Analyse-Chip wieder ausgleicht. Am Ausgang steht je nach verwendetem D/A-Umsetzer entweder ein linearer 16-Bit-Code oder ein logarithmisch komprimierter Code (A-Law) zur Verfügung.

5 Entwurf eines Vocoders

Mit diesem Chip ist es möglich, einen Duplex-LPC-Vocoder (*Bild 7*) auf einer Europlatine doppelter Größe ($23 \times 16\ cm^2$) aufzubauen. Vierfach-Operationsverstärker und Einchip-Codecs mit Filtern halten die Chipan-

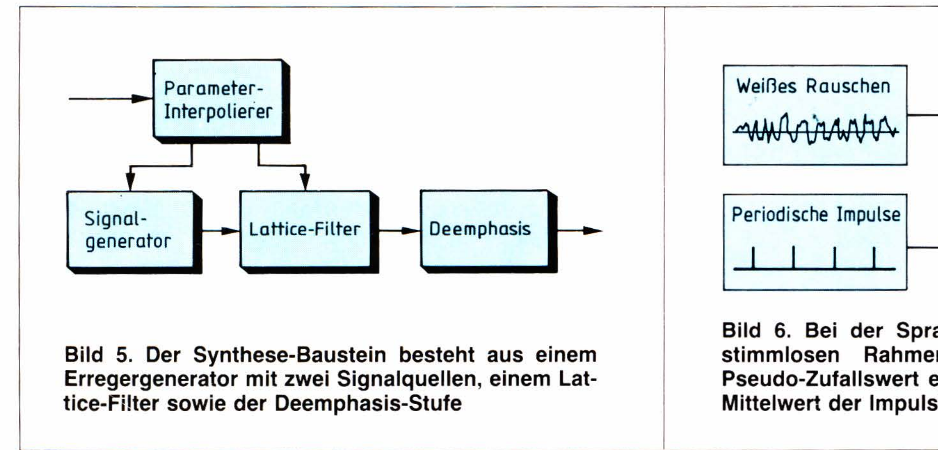


Bild 5. Der Synthese-Baustein besteht aus einem Erregergenerator mit zwei Signalquellen, einem Lattice-Filter sowie der Deemphasis-Stufe

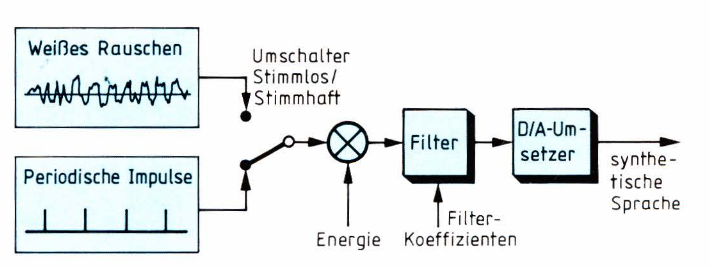


Bild 6. Bei der Sprachsignalgenerierung wird während einer stimmlosen Rahmenperiode das Lattice-Filter mit einem Pseudo-Zufallswert erregt. Bei einem stimmhaften Laut ist der Mittelwert der Impulsfolge Null

Einchip-Computer µPD 7720 führt 4 Mio. Multiplikations-Additionszyklen in einer Sekunde aus

Funktion

Beim µPD 7720 SPI, der in Hochgeschwindigkeits-NMOS-Technologie gefertigt wird, handelt es sich um einen vollständigen 16-Bit-Mikrocomputer, der in einem 28poligen DIP-Gehäuse untergebracht ist.

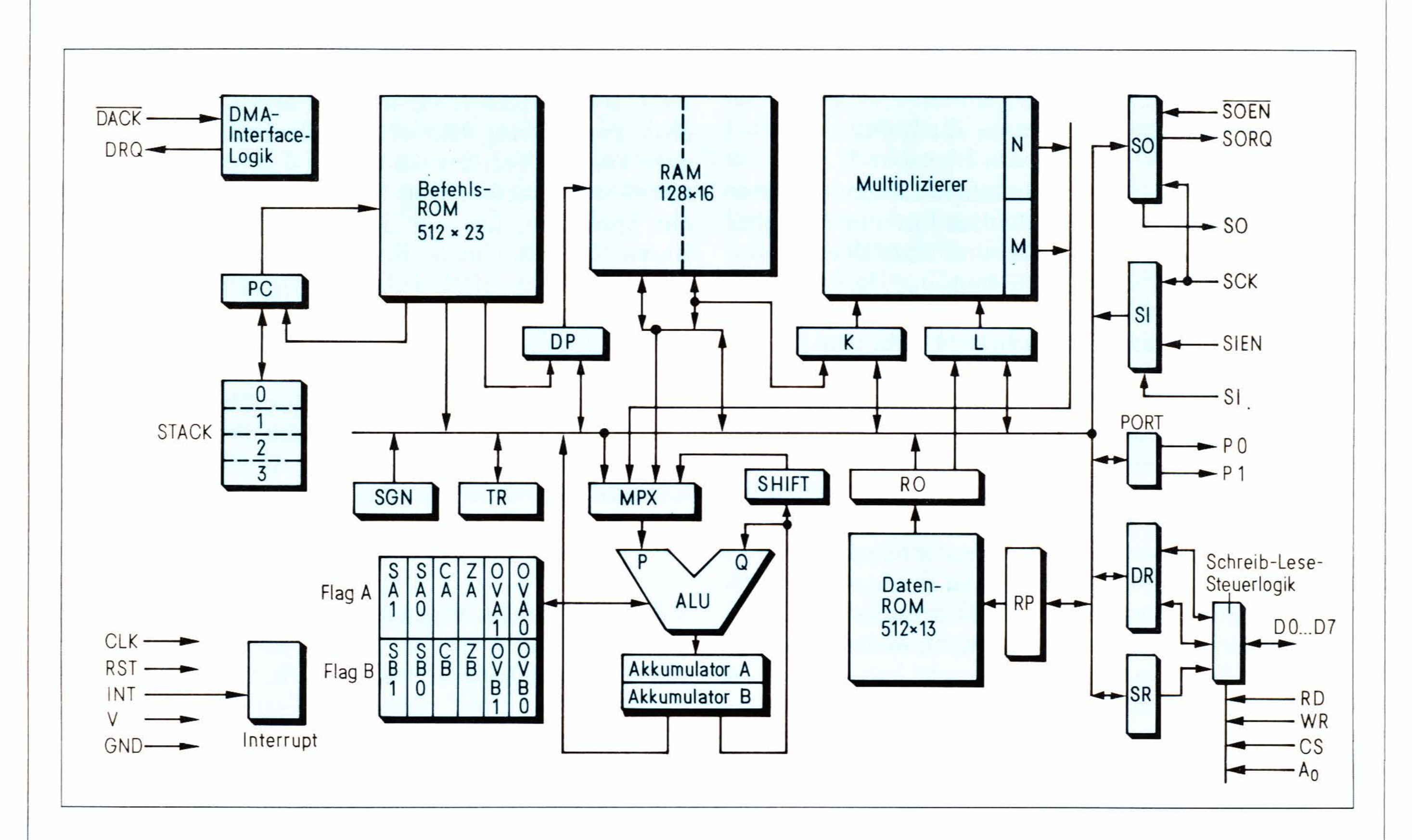
ROM-Kapazität für Programm und Koeffizienten ist vorhanden, während der auf dem Chip vorhandene RAM-Bereich zum vorübergehenden Speichern von Daten, Koeffizienten und Ergebnissen benutzt werden kann.

Rechnungen werden von einer 16-Bit-Arithmetik/Logik-Einheit und einem separaten parallelen 16x16-Multiplizierer ausgeführt. Diese Kombination erlaubt die Berechnung von „Summe von Produkten" in einem einzigen Befehlszyklus von 250 ns. Zusätzlich sorgt jeder OP-Befehl für eine Reihe von Datenbewegungsoperationen, um den Gesamtdurchsatz noch weiter zu erhöhen.

Die seriellen und die parallele E/A sind per Software konfigurierbar zu 8 bzw. 16 Bit Wortlänge. Zwei serielle E/A-Ports stehen als Interface zu A/D-, D/A-Umsetzern, Codecs oder anderen seriell orientierten Funktionselementen zur Verfügung, während ein paralleler 8-Bit-Bus sowohl Daten als auch Zustandsinformation an übliche µPs weiterreichen kann. Handshaking-Signale und ein DMA-Interface erlauben es, den SPI sowohl als hochentwickelte, programmierbare Peripherie als auch als selbständigen Mikrocomputer anzuwenden.

Speicher

Es stehen drei Speicher zur Verfügung: Programm-ROM, Daten-ROM und Daten-RAM. Das Programm-ROM (512x23 Bit) wird über einen 9-Bit-Programmzähler adressiert, der auch durch externe Reset-Befehle und Interrupts oder Call-, Jump- bzw. Return-Anweisungen modifiziert werden kann.



Das Daten-ROM (512x13 Bit) wird ebenfalls durch einen 9-Bit-Pointer adressiert, der als Teil eines OP-Befehls modifiziert werden kann, so daß der nächste Wert für den nächsten Befehl zur Verfügung steht. Dieser Speicher ist besonders zur Ablage der benötigten Koeffizienten, Umwandlungstabellen und Konstanten für alle Ihre Verarbeitungsaufgaben geeignet.

Das Dual-Port-Daten-RAM besteht aus 128 Worten mit jeweils 16 Bit und wird von einem 7-Bit-Datenpointer (DP) adressiert. Dieser hat weitgehende Adressierungsmöglichkeiten simultan zu den arithmetischen Anweisungen, so daß keine zusätzliche Zeit benötigt wird, um Adressen anzuwählen oder zu modifizieren.

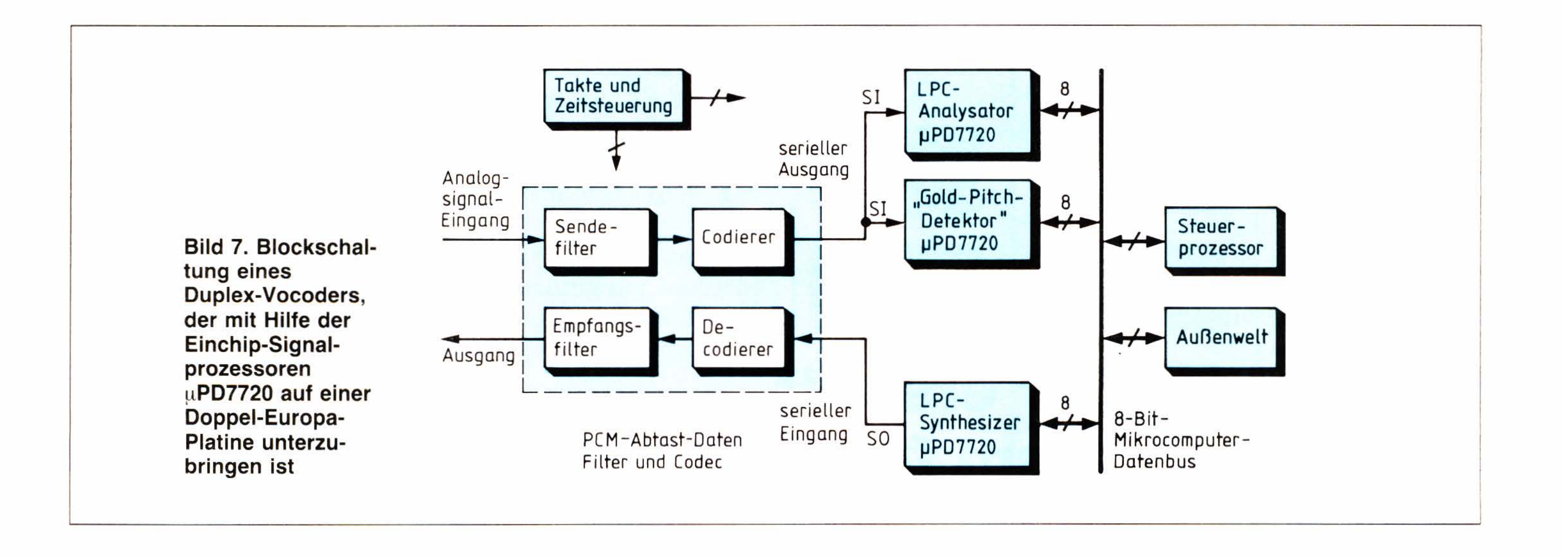


Bild 7. Blockschaltung eines Duplex-Vocoders, der mit Hilfe der Einchip-Signalprozessoren µPD7720 auf einer Doppel-Europa-Platine unterzubringen ist

zahl der analogen Zwischenstufen niedrig. Jeder Allzweck-µP ist als Steuerprozessor geeignet. Für die Kommunikation über ein Modem ist ein serieller, synchroner Duplex-Kanal mit Leitungstreibern und Leitungsempfängern erforderlich.

Literatur

[1] *LeRoux, I. J., Gueguen, C.*: A Fixed Point Computation of Partial Correlation Coefficients. IEEE Trans. ASSP, Vol. 25, S. 257...259, June 1977.

[2] *Tohkura, Y., Itakura, F., Hashimoto, S.*: Spectral Smoothing Techniques in PARCOR Speech Analysis. IEEE Trans. ASSP, Vol 26, S. 587...596, Dec. 1978.

[3] *Gold, B., Rabiner, L. R.*: Parallel Processing Techniques for Estimating Pitch Periods of Speech in the Time Domain. J. Acoust. Soc. Am. Vol. 46, No. 2, Pt. 2, S. 442...448, Aug. 1969.

[4] *Atal, B. S., Rabiner, L. R.*: A Pattern Recognition Approach to Voiced-Unvoiced-Sillence Classification with Applications to Speech Recognition. IEEE Trans. ASSP, Vol. 24, S. 201...212, Juni 1976.

[5] *Kawakami, Y., Nishitani, T., et. al.*: A single-chip digital signal processor for voiceband applications. ISSCC Dig. Tech. Papers, S. 40...41, 1980.

[6] *Hedelin, P.*: RELP-Vocoding with uniform and non-uniform down-sampling, ICASSP-1983.

[7] *Feldman, J. A., Hofstetter, E. M.*: A Compact, Flexible LPC Vocoder based on a Commercial Signal Processing Microcomputer. IEEE Proc. Int. Conf. ASSP 1982.

[8] *Feldman, J. A.*: A Compact Channel Vocoder using Commercial Devices. IEEE Proc. Int. Conf. AASP 1982.

[9] *Hertel, J., Mattukat, S.*: Erstellung eines 10-Kanal-Vocoder-Systems. Diplomarbeit, FH Bochum, März 1982.

LPC – wie und warum?

Im Zuge der fortschreitenden Umstellung der Kommunikationssysteme auf digitale Arbeitsverfahren wurde eine große Zahl von Sprachcodierungssystemen (Vocoder) entwickelt. Jede Methode ist ein Kompromiß aus Faktoren wie Signalqualität, Übertragungsrate und Codiererkomplexität.

Vocoder können in zwei Gruppen eingeteilt werden: Wellenformcodierer und parametrische Codierer. Wie der Name sagt, versucht man mit dem Wellenformcodierer die Signalform zu rekonstruieren, während parametrische Codierer nur versuchen, das Eingangsspektrum zu rekonstruieren. „LPC" (Linear Predictive Coding) ist eine Form der parametrischen Codierung. Der Hauptvorteil liegt in der geringen Bandbreite. Um ein Sprachsignal mit einer Bandbreite von 3,5 kHz zu reproduzieren, braucht LPC lediglich 2,4 kBit/s, während Standard-log-PCM 64 kBit/s benötigt.

LPC basiert auf zwei Voraussetzungen:

1. Das Sprachsignal verändert sich langsam. Seine Charakteristica sind über einen Zeitraum von etwa 10...20 ms annähernd konstant. Dieser Zeitraum heißt „Frameperiode" (Rahmenperiode).
2. Innerhalb eines Rahmens kann das Sprachsignal durch das Ausgangssignal eines variablen Filters angenähert werden, das durch eine Schallquelle erregt wird. Die Schallquelle produziert entweder weißes Rauschen oder eine Impulsfolge (*Bild 6*).

Das Filter ist ein Modell des Sprachtaktes; die Impulse entsprechen der Sprachlage (Pitch) von stimmhaften Lauten (Vokale) und die Rauschquelle entspricht dem Rauschen, das von dem sich zusammenziehenden Sprachtakt bei stimmlosen Lauten erzeugt wird (Konsonanten).

Um Sprache zu reproduzieren, ist ein Filter der Ordnung 10 oder größer nötig. Die Rahmenperiode ist dabei typischerweise 20 ms.

Der LPC-Analysator berechnet die optimalen Filterkoeffizienten für jeden Rahmen und klassifiziert diesen außerdem als stimmlos oder stimmhaft. In stimmhaften Rahmen wird auch die Stimmlage (Pitchperiode) geschätzt. Der LPC-Synthesizer bildet die Schallquellen-Filter-Struktur mit den vom Analysator bestimmten Filterkoeffizienten und der Erregung nach.

Dipl.-Ing. Wolfgang Loges

Codegenerator erstellt Reglerprogramm für TMS320

Den Signalprozessor TMS320 kann man zur Zeit noch nicht in einer höheren Programmiersprache, wie z.B. Pascal, programmieren. Aus diesem Grund ist man gezwungen, die Programmierung in Assemblersprache vorzunehmen. Dies ist allerdings unpraktisch, wenn sich in der Laborphase die Daten des Reglers noch ändern, da dies oft zu einem neuen Assemblerprogramm führt. Aus diesem Grund wurde ein Codegenerator in UCSD-Pascal entwickelt, der für Mehrgrößen-Regler automatisch das Assemblerprogramm für den TMS320 erstellt. Er läuft auf den Tischrechnern HP9836 und Kontron PSI80.

1 Nutzen des Codegenerators

Soll der Signalprozessor TMS320 als universelles Reglersystem im Labor Verwendung finden [1], ist es von Vorteil, wenn der Regler sich durch schnelles Umprogrammieren an verschiedene Regelungsaufgaben anpassen läßt. Die Programmierung von „Hand" in Assemblersprache ist sehr mühsam und fehleranfällig, wobei viel Zeit investiert werden muß, bis der jeweilige Regler läuft. In der Erprobungsphase eines Reglers, in der sich noch Parameter ändern können, ist die Handprogrammierung in Assemblersprache ebenfalls zeitraubend und fehleranfällig. Durch die Änderung einiger Parameter kann es sich z.B. ergeben, daß sich die Anzahl der Koeffizienten erhöht. Ergibt sich nun, daß die Anzahl der Koeffizienten die Zahl der verfügbaren Speicherplätze im Daten-RAM übersteigt, ist man gezwungen, größere Teile des Assemblerprogramms zu ändern. Eine Änderung des Assemblerprogramms ist ebenfalls notwendig, wenn sich herausstellt, daß in Festkommaarithmetik die Genauigkeit von 16 Bit, mit denen die Daten und Koeffizienten dargestellt werden, nicht ausreicht. In diesem Fall muß dann mit doppelter Genauigkeit gerechnet werden. Dabei besteht die Möglichkeit, wahlweise nur die Daten bzw. Koeffizienten, oder beide in doppelter Genauigkeit zu verwenden. Für jede Variante ist ein gesondertes Assemblerprogramm zu erstellen. Diese Möglichkeiten lassen sich im Labor nur dann sinnvoll nutzen, wenn die Assemblerprogramme durch einen Codegenerator automatisch erstellt werden.

Bild 1. Zeitlicher Ablauf des Regelalgorithmus

2 Abarbeitung des Regleralgorithmus

Dem Assemblerprogramm, das vom Codegenerator erstellt wird, liegt folgender Algorithmus zugrunde:

$$\underline{x}_{k+1} = \underline{A}\,\underline{x}_k + \underline{B}\,\underline{u}_k + \underline{f}_x(\underline{x}_k,\underline{u}_k,k)$$
$$\underline{y}_k = \underline{C}\,\underline{x}_k + \underline{D}\,\underline{u}_k + \underline{f}_y(\underline{x}_k,\underline{u}_k,k)$$

Dabei sind $\underline{A}$, $\underline{B}$, $\underline{C}$ und $\underline{D}$ die Matrizen des linearen Reglers in Zustandsform. Die Vektoren $\underline{f}_x$ und $\underline{f}_y$ enthalten die nichtlinearen Terme des Reglers. Sind keine nichtlinearen Terme vorhanden, entfallen die Vektoren $\underline{f}_x$ und $\underline{f}_y$. Das Abarbeiten des Algorithmus geschieht in der Weise, daß nach der Abtastung der Meßgrößen $\underline{u}_k$ nur noch das $\underline{D}\,\underline{u}_k + \underline{f}_y'$ berechnet wird, wobei $\underline{f}_y'$ die nichtlinearen Terme von $\underline{f}_y$ enthält, die mit den aktuellen Meßgrößen $\underline{u}_k$ verknüpft sind. Zu den Ergebnissen von $\underline{D}\,\underline{u}_k + \underline{f}_y'$ addiert man die im vorhergehenden Abtastschritt ermittelten Werte von $\underline{y}' = \underline{C}\,\underline{x}_k + \underline{f}_y'''$. Das

$\underline{f}_y$''' enthält dabei die nichtlinearen Terme, die nicht mit den aktuellen Meßgrößen $\underline{u}_k$ verknüpft sind. Die Ergebnisse lassen sich noch über die nichtlinearen Terme $\underline{f}_y$''' bewerten, bevor sie als Stellgrößen $\underline{y}_k$ ausgegeben werden. Nach der Ausgabe der Stellgrößen werden die Werte $\underline{x}_{k+1} = \underline{A}\ \underline{x}_k + \underline{B}\ \underline{u}_k + \underline{f}_x$ und $y' = \underline{C}\ \underline{x}_k + \underline{f}_y'''$ bestimmt.

Durch diese Reihenfolge der Berechnung erreicht man, daß die Zeit zwischen der Abtastung der Meßgrößen $\underline{u}_k$ und der Ausgabe der Stellgrößen $\underline{y}_k$ möglichst klein ist, da sie sich als Totzeit im Regler bemerkbar macht.

3 Struktur des Codegenerators

3.1 *Aufbereitung der Koeffizienten des linearen Reglers*

Die Koeffizienten des linearen Reglers, d.h. die Elemente der Matrizen $\underline{A}$, $\underline{B}$, $\underline{C}$ und $\underline{D}$, die von einer Reglerentwurfs-Software geliefert werden, stehen als Realzahlen in einer Datei auf der Diskette. Neben den Koeffizienten sind noch Angaben über die Ordnung des Systems und die Anzahl der Ein- und Ausgänge des linearen Reglers enthalten. Da der Signalprozessor einen Zahlenbereich von –1,0 bis +1,0 hat, bei Linkskommadarstellung, können nur die Koeffizienten, die in diesen Zahlenbereich fallen, direkt in eine Binärzahl umgewandelt werden. Bei den Koeffizienten, die noch einen ganzzahligen Anteil besitzen, ist dieser abzutrennen, da er vom Codegenerator bei der Skalarproduktbildung gesondert berücksichtigt werden muß. Der gebrochene Anteil läßt sich wie bei den anderen Koeffizienten in eine Binärzahl umwandeln. Vor Ausführung der Umwandlung besteht die Möglichkeit, die Koeffizienten der Skalarprodukte so zu skalieren, daß im ungünstigsten Fall ein bestimmter Wert nicht überschritten wird. Diesen Wert wählt man günstigerweise so, daß er dem maximal darstellbaren Zahlenbereich des Akkumulators im TMS 320 (–2,0...+2,0) entspricht. Für den ungünstigsten Fall geht man davon aus, daß die variablen Daten betragsmäßig maximal 1,0 werden können, und somit die Beträge der Koeffizienten jeweils eines Skalarproduktes aufzuaddieren sind. Überschreitet die Summe den vorgegebenen Zahlenbereich, werden die Koeffizienten so lange durch zwei dividiert, bis die Summe den Zahlenbereich nicht mehr überschreitet. Dabei bedeutet jede Division durch zwei im Signalprozessor eine Verschiebung um eine Stelle nach rechts. Diese vorgenommene Skalierung muß nach der Berechnung des jeweiligen Skalarproduktes wieder rückgängig gemacht werden, um die ursprüngliche Stellenwertigkeit wieder zu erlangen. Ziel dieser Skalierung ist es, zu verhindern, daß während der Akkumulation der Partialsummen eines Skalarproduktes ein Überlauf auftritt. Dieser könnte sich bei der Regelung ungünstig auswirken. Ist die gewünschte Skalierung durchgeführt, beginnt man mit der Umwandlung der Realzahlen in eine Binärzahl, wobei noch unterschieden wird, mit welcher Genauigkeit gerechnet werden soll. Es stehen drei Möglichkeiten zur Auswahl:

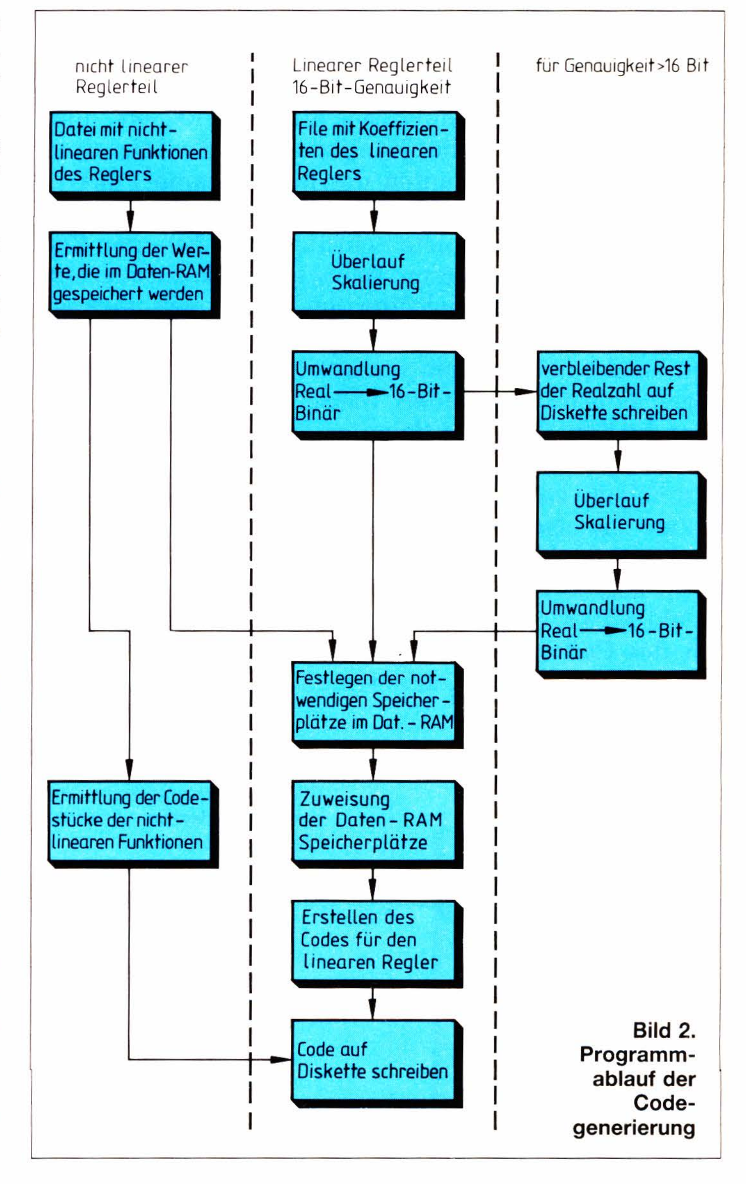


Bild 2. Programmablauf der Codegenerierung

a) Daten und Koeffizienten in 16 Bit
b) Daten in 16 Bit Koeffizienten in 32 Bit
c) Daten und Koeffizienten in 32 Bit

Für die Fälle b) und c) wird die Umwandlung in zwei Schritten vorgenommen. Im ersten Schritt werden die höherwertigen 16 Bit ermittelt. Dies entspricht dem Anteil der Realzahl, der durch die Potenzen $2^{-1}...2^{-15}$ darzustellen ist. Der sich ergebende Rest der Realzahl kommt in eine temporäre Datei zur Zwischenspeicherung. Im zweiten Schritt dient dieser Rest für die Ermittlung der niederwertigen 16 Bit, die dem Anteil der Realzahl entspricht, der mit den Potenzen $2^{-16}...2^{-30}$ dargestellt wird (*Bild 2*).

3.2 *Zuordnung der Speicherplätze im Daten-RAM*

Bevor man mit der Codierung für das Reglerprogramm beginnen kann, muß man feststellen, wieviele Daten-RAM-Speicherplätze benötigt werden. Dies ist erforder-

lich, da man mit den 144 Daten-RAM-Plätzen des TMS 320 schnell an Grenzen stößt. Die benötigte Anzahl an Speicherplätzen bestimmen die Elemente $\underline{u}_k$, $\underline{x}_k$, $\underline{x}_{k+1}$, $\underline{y}_k$ und den Koeffizienten des linearen Reglers, und, soweit vorhanden, die Koeffizienten, Zwischenspeicherplätze und Konstanten der nichtlinearen Funktionen des Reglers (*Bild 3*). Reicht der Speicherplatz nicht, muß man durch geeignete Schritte versuchen, den Regler trotzdem zu realisieren. Dabei ist davon auszugehen, daß die Anteile der nichtlinearen Funktionen unverändert im Daten-RAM gespeichert werden müssen, da die Codestücke für die nichtlinearen Funktionen fertig vorliegen und nicht geändert werden sollen. Aus diesem Grund müssen die Koeffizienten des linearen Reglers so modifiziert sein, daß der Speicherplatz des Daten-RAMs ausreichend ist. Hierfür bieten sich mehrere Möglichkeiten an. Die erste Möglichkeit ist, nur die Koeffizienten, die kleiner −4096 und größer 4095 sind, im Daten-RAM zu speichern, während die anderen Koeffizienten in den Befehl MPYK eingebunden werden und somit Bestandteil des Codes im Programmspeicher werden. Reicht diese Aufteilung der Koeffizienten nicht aus, müssen sie in einen 13-Bit- und einen 16-Bit-Anteil aufgespalten werden. Bei dieser Aufteilung sind nur noch 15 Speicherplätze für den 16-Bit-Anteil im Daten-RAM nötig. Der 13-Bit-Anteil wird wieder in den Befehl MPYK eingebunden [2]. Liegt die Aufteilung des Daten-RAMs fest, kann man mit der Codeerzeugung für das Reglerprogramm beginnen.

3.3 Assemblerprogrammerstellung

Der Codegenerator ist so organisiert, daß er zuerst das Codestück für die symbolische Zuweisung der Speicherplätze des Daten-RAMs erstellt. Hiernach folgt das Codestück für die Daten-RAM-Belegung und das Codestück für die Datenübernahme aus dem Programmspeicher in das Daten-RAM (*Bild 4* und *6*). Nach den Eingabebefehlen für die Eingangsgrößen $\underline{u}_k$ folgt das Codestück für die Teilskalarprodukte $\underline{D}\ \underline{u}_k$, gefolgt von den Outputbefehlen für die Stellgrößen $\underline{y}_k$. Danach folgen die Codestücke für die Skalarprodukte $\underline{x}_{k+1}$ und $\underline{C}\ \underline{x}_k$. Dabei ist der Codegenerator so konzipiert, daß er für jedes Skalarprodukt ein separates Codestück anlegt, dessen Länge sich nach der Anzahl der Koeffizienten richtet. Auf eine Schleifenstruktur ist aus Gründen der optimalen Zeitausnutzung verzichtet worden. Den ganzzahligen Anteil eines Koeffizienten berücksichtigt der Codegenerator in dem jeweiligen Skalarprodukt durch eine entsprechende Befehlskonstruktion. Beim Codeaufbau für die Skalarprodukte entscheidet der Codegenerator selbständig, mit welchen Befehlen die Partialsummen berechnet werden müssen, d.h. er berücksichtigt eine eventuell notwendig gewordene Aufspaltung der Koeffizienten. Sind die Koeffizienten eines Skalarproduktes skaliert worden, erstellt der Codegenerator für diese Skalarprodukte auch die notwendigen Codestücke für die Rückskalierung und Überlauf-Untersuchung, die dem jeweiligen Codestück für das Skalarprodukt hinzugefügt wird. Sind auch nichtlineare Funktionen des Reglers zu berücksichtigen, so fügt der Codegenerator an den entsprechenden Stellen des Codes für den

Bild 3. Speicherbelegung im Daten-RAM

Bild 4. Programmaufbau des generierten Assemblerprogramms

Symbolische Adressen der variablen Daten

```
U(K)1:EQU;0!
X(K+1)1:EQU;1!
X(K+1)2:EQU;3!
X(K+1)3:EQU;5!
X(K)1:EQU;2!
X(K)2:EQU;4!
X(K)3:EQU;6!
Y(K)1:EQU;7!
Y(K)L1:EQU;8!
```

Symbolische Adressen der Koeffizienten

```
A1 1:EQU;9!   ADRESSE
A1 2:EQU;10!   ADRESSE
A2 1:EQU;11!   ADRESSE
A2 2:EQU;12!   ADRESSE
A3 3:EQU;13!   ADRESSE
B1 1:EQU;14!   ADRESSE
B2 1:EQU;15!   ADRESSE
B3 1:EQU;16!   ADRESSE
C1 1:EQU;17!   ADRESSE
C1 2:EQU;18!   ADRESSE
C1 3:EQU;19!   ADRESSE
D1 1:EQU;20!   ADRESSE
        :B;JUMPA!
```

Speicherplätze für variable Daten

```
        :DW;0!
        :DW;0!
        :DW;0!
        :DW;0!
        :DW;0!
        :DW;0!
        :DW;0!
        :DW;0!
        :DW;0!
```

Koeffizienten

```
        :DW;12283!
        :DW;10496!
        :DW;-10496!
        :DW;12283!
        :DW;12054!
        :DW;-4462!
        :DW;-6806!
        :DW;4947!
        :DW;10240!
        :DW;10240!
        :DW;22845!
        :DW;-879!
   ADR:DW;0!
```

Übernahme der Daten aus dem Programm-RAM ins interne Daten'RAM

```
 JUMPA:LARK,1;ADR!
        :LARK,0;20!
        :LARK,0;127!
        :LDPK;0!
 JUMPB:SAR,1;0!
        :ZALS;0!
        :MAR;*1!
        :MAR;*-!
        :MAR;*0!
        :TBLR;*0!
        :BANZ;JUMPB!
        :SOVM!
        :B;JUMPC!
  ADR1:DW;32767!
  ADR2:DW;-32768!
  ADR3:DW;0!
  ADR4:DW;-1!
 JUMPC:LDPK;1!
        :LACK;ADR1!
        :TBLR;0!
        :LACK;ADR2!
        :TBLR;1!
        :LACK;ADR3!
        :TBLR;2!
        :LACK;ADR4!
        :TBLR;3!
        :LDPK;0!
```

Input $\underline{u}_k$

```
JUMPE:BIOZ;JUMPE!
        :IN,0;U(K)1!
        :ZAC!
        :LT;U(K)1!
        :MPY;D1 1!
        :APAC!
        :ADDH;Y(K)1!
```

$\underline{y}_k = \underline{y}' + \underline{D}\ \underline{u}_k$

Rückskalierung und Overflow-Abfrage von $\underline{y}_k$

```
        :ADDS;Y(K)L1!
              :BGEZ;JAY(K)1!
              :LDPK;1!
              :SACH,0;12!
              :SACL;13!
              :LAC,8;12!
              :SACH,0;14!
              :ZALS;14!
              :XOR;3!
              :BZ;JBY(K)1!
              :ZALS;1!
              :LDPK;0!
              :SACL;Y(K)1!
              :B;JCY(K)1!
JAY(K)1        :LDPK;1!
              :SACH,0;12!
              :SACL;13!
              :LAC,8;12!
              :SACH,0;14!
              :ZALS;14!
              :XOR;2!
              :BZ;JBY(K)1!
              :ZALS;0!
              :LDPK;0!
              :SACL;Y(K)1!
               :B;JCY(K)1!
JBY(K)1           :LAC,7;13!
              :SACH,0;13!
              :ZALS;13!
              :AND;0!
              :SACL;13!
              :LAC,7;12!
              :SACL;12!
              :ZALH;12!
              :ADD,8;13!
              :LDPK;0!
              :SACH,0;Y(K)1!
JCY(K)1        :LDPK;0!
```

Output $\underline{y}_k$

```
         :OUT,1;Y(K)1!
```

$\underline{x}_{k+1} \rightarrow \underline{x}_k$

```
        :DMOV;X(K+1)1
        :DMOV;X(K+1)2!
        :DMOV;X(K+1)3!
```

x-update

$\underline{x}_{k+1} = \underline{A}\ \underline{x}_k + \underline{B}\ \underline{u}_k$

```
         :ZAC!
         :LT;X(K)1!
         :MPY;A1 1!
         :LTA;X(K)2!
         :MPY;A1 2!
         :LTA;U(K)1!
         :MPY;B1 1!
         :APAC!
         :SACH,1;X(K+1)1!
         :ZAC!
         :LT;X(K)1!
         :MPY;A2 1!
         :LTA;X(K)2!
         :MPY;A2 2!
         :LTA;U(K)1!
         :MPY;B2 1!
         :APAC!
         :SACH,1;X(K+1)2!
         :ZAC!
         :LT;X(K)3!
         :MPY;A3 3!
         :LTA;U(K)1!
         :MPY;B3 1!
         :APAC!
         :SACH,1;X(K+1)3!
         :ZAC!
         :LT;X(K+1)1!
         :MPY;C1 1!
         :LTA;X(K+1)2!
         :MPY;C1 2!
         :LTA;X(K+1)3!
         :MPY;C1 3!
         :APAC!
         :SACH,0;Y(K)1!
         :SACL;Y(K)L1!
        :B;JUMPE!
```

Bild 6. Generiertes Reglerprogramm (Regler 3. Ordnung, 1 Eingang, 1 Ausgang)

linearen Regler Sprungmarken ein (Bild 4), die auf den Code der nichtlinearen Funktionen verweisen. Die Codestücke werden vom Codegenerator an das Ende des Codes für den linearen Regler angehängt. Diese Codestücke werden nicht vom Codegenerator erzeugt, sondern aus einer Bibliothek bezogen, die auf einer Diskette steht.

4 Aufbau der nichtlinearen Funktionen

Die Vektoren $\underline{f}_x$ und $\underline{f}_y$ enthalten die nichtlinearen Funktionen des Reglers. Sie stehen getrennt von den Koeffizienten des linearen Reglers auf einer Diskette. Die Elemente der Vektoren $\underline{f}_x$ und $\underline{f}_y$ haben die folgende Form

f_x,n, f_y,n : G[a,b,c,...].

G steht hier für einen Funktionsnamen, unter dem der Codegenerator in einer Bibliothek ein fertiges Codestück für die Funktion findet. Die Buchstaben in der eckigen Klammer geben die Variablen der Funktion an. Ein Beispiel zeigt dies:

x(k+1)1 : QU [x(k+1)1, u(k)2]

Das QU steht hier für die Funktion $e = a + b^2$, wobei in diesem Beispiel a=x(k+1)1 und b=u(k)2 ist. In Worten ausgedrückt heißt dies, daß zu dem Wert x(k+1)1 der quadrierte Eingangswert u(k)2 addiert werden soll. Das Ergebnis ist dann wieder auf dem Speicherplatz von x(k+1)1 zu speichern. Die Syntax, in der die Funktionen als Datei zu erstellen sind, zeigt *Bild 5*. Diese Datei gliedert sich in zwei Teile, dem Vereinbarungs- und dem Codeteil. Im Vereinbarungsteil stehen die benutzten Konstanten, die benötigten Zwischenspeicherplätze, verwendete Namen für Sprungadressen und die symbolischen Namen für die Variablen. Werden einzelne Teile im Vereinbarungsteil nicht benötigt, bleiben diese Felder leer.

```
*        ________
         Namen und Wert der Konstanten. Die Namen
         sind an den Funktionsnamen gekoppelt
* *      ________
         Namen der benötigten Zwischenspeicherplätze
* * *    ________
         Verwendete symbolische Namen
* * * *  ________
         Namen für Sprungadressen (sind an den
         Funktionsnamen gekoppelt)
* * * * *  ________
         Anzahl der benötigten Instruktionsworte
* * * * * *  ________
         Codestück der nichtlinearen Funktion
* * * * * * *  ________
```

Bild 5. Filestruktur für die nichtlinearen Funktionen

Für die oben angegebene Funktion (x(k+1)1 : QU [x(k+1)1,u(k)2]) hat das File die folgende Form:

```
Filename: QU
*
* *
* * *
AS
BS
CS
* * * *
* * * * *
TI:=5
* * * * * *
LAC,15;BS !
LT;CS     !
MPY;CS    !
APAC      !
SACH,1;AS!
* * * * * * *
```

Die symbolischen Namen AS, BS, CS korrespondieren mit den Bezeichnern im Funktionsaufruf, d.h. in diesem Beispiel:

AS ≙ x(k+1)1 neu
BS ≙ x(k+1)1 alt
CS ≙ u(k)2

Während der Code an das Programm des linearen Reglers angebunden wird, sucht der Codegenerator den Code nach den symbolischen Namen ab, und fügt an den Stellen entsprechend der oben angegebenen Zuordnung die entsprechenden Werte ein.

Literatur

[1] *Loges, W.*: Schneller digitaler Regler mit Signalprozessor. ELEKTRONIK 1983, H. 19, S. 51...54.
[2] *Loges, W.*: Regelsysteme höherer Ordnung mit dem Signalprozessor TMS320. ELEKTRONIK 1983, H. 25, S. 53...55.
[3] *Hanselmann, H., Loges, W.*: Realisierung schneller digitaler Regler hoher Ordnung mit Signalprozessoren. Regelungstechnik 1983, H. 10, S. 330...337.
[4] *Hanselmann, H.*: Tischrechner programmiert Signalprozessor als digitalen Mehrgrößenregler. ELEKTRONIK 1982, H. 21, S. 134...138.

Dipl.-Ing. Wolfgang Loges

Schneller digitaler Regler mit Signalprozessor

Signalprozessoren eignen sich aufgrund ihrer speziellen Merkmale auch zur Realisierung von schnellen digitalen Mehrgrößen-Reglern. Mit dem hier verwendeten Signalprozessor TMS 320 lassen sich Abtastraten erreichen, die mit herkömmlichen Mikroprozessoren nicht zu erzielen sind. Das hier vorgestellte System ist so konzipiert, daß es auch andere Aufgaben der digitalen Signalverarbeitung lösen kann.

Zentrale Rechenoperation in der digitalen Regelungs- und Filtertechnik ist die Berechnung von Skalarprodukten. Bei der Bildung des Skalarproduktes müssen Multiplikationen und Additionen ausgeführt werden. So kann z. B. bei der Realisierung eines Reglers 10. Ordnung mit fünf Eingängen (Meßgrößen) und einem Ausgang (Steuergröße) die Berechnung von elf Skalarprodukten erforderlich sein. Je nach Aufbereitung der Reglerdaten bedeutet dies zwischen 75 und 165 Multiplikationen und 64 bis 154 Additionen pro Abtastschritt. Um hier auf kurze Abtastzeiten zu kommen, benötigt man ein System, das die Multiplikation und Addition schnell ausführen kann. Hier bietet sich der Signalprozessor TMS320 von Texas Instruments an, der über einen 16×16-Bit-Hardware-Multiplizierer auf dem Chip verfügt. Pro Multiplikation benötigt der Baustein nur einen Maschinenzyklus, was bei maximaler Taktrate des TMS320 einer Zeit von 200 ns entspricht. Die Akkumulation der Produkte erfolgt mit einer Breite von 32 Bit. Da der TMS320 – obwohl als „Signalprozessor" bezeichnet – in Strukturen und Befehlssatz Merkmale eines Universal-Mikroprozessors aufweist, ist er auch z. B. für Aufgaben der adaptiven Regelung geeignet, wo über die obengenannte Skalarproduktbildung hinaus auch noch andere Operationen zu realisieren sind.

1 Systemstruktur

Beim Systemaufbau wurde auf eine modulare Struktur Wert gelegt. Hierdurch bietet sich die Möglichkeit der späteren Erweiterung und Anpassung an wechselnde Anforderungen. Das System besteht aus den Modulen Prozessorkarte mit dem TMS320, Interfacekarte, Steuerkarte und den ADU/DAU-Subsystemen (*Bild 1*). Für die ADU- und DAU-Subsysteme stehen zur Zeit sechs Steckplätze zur Verfügung, die beliebig bestückt werden können. Dabei enthält das ADU-Subsystem einen Analog/Digital-Umsetzer und ein DAU-Subsystem zwei Digital/Analog-Umsetzer. Über die Interfacekarte wird das Programm, das auf einem Gastrechner erstellt wurde, in den Programmspeicher des TMS320 geladen. Dazu kann der Prozessor angehalten und vom Adreß-, Daten- und Steuerbus getrennt werden. Weiterhin hat man vom Gastrechner Zugriff zur Steuerkarte, um dort zu Beginn definierte Zustände zu erzeugen. Auch zu den ADU-DAU-Subsystemen hat man mit dem Gastrechner Zugriff. Die Steuerkarte sorgt für den richtigen zeitlichen Ablauf der Datenübertragung vom ADU-Subsystem zum Prozessor und der Übertragung der Prozessordaten in das DAU-Subsystem.

2 Hardware

2.1 Prozessorkarte

Diese Karte enthält den Prozessor TMS320, den Programmspeicher, bestehend aus 4-K×4-Bit-RAMs mit

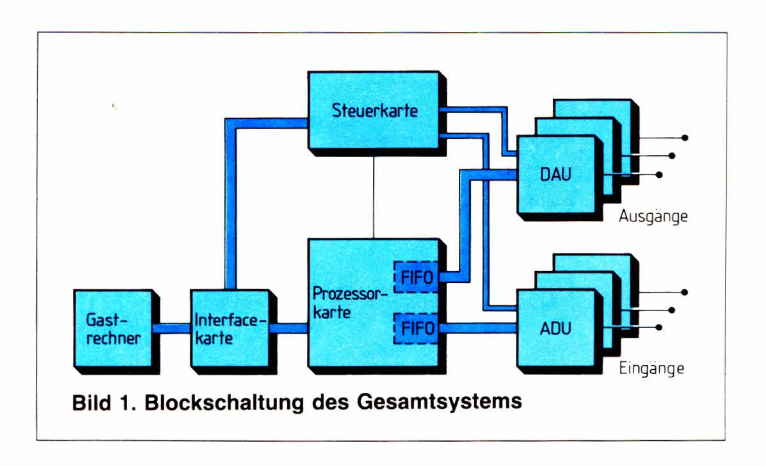


Bild 1. Blockschaltung des Gesamtsystems

50 ns Zugriffszeit, FIFO-Speicher und Tristate-Buffer als Buskoppler. Mit Hilfe dieser Tristate-Buffer kann man Systemgruppen vom Bussystem abkoppeln. Hierdurch bietet sich die Möglichkeit, vom Gastrechner aus über die Interfacekarte auf Programm- und FIFO-Speicher zuzugreifen. Die Ausgänge der Tristate-Buffer werden ebenfalls über die Interfacekarte vom Gastrechner aus gesteuert. Es ist hierüber möglich, das Programm und auch Daten auf dem Gastrechner vorzubereiten und dann in den Programmspeicher des TMS320 zu laden. Dazu wird der Prozessor durch die Tristate-Buffer vom Adreß- und Steuerbus getrennt und vom Datenbus, indem man ihn in die Reset-Stellung bringt. Nach Beendigung der Datenübertragung wird der Gastrechner durch Tristate-Buffer von der Prozessorkarte getrennt und der TMS320 wieder an die Bussysteme angeschlossen, so daß er selbständig laufen kann. Es besteht aber jederzeit die Möglichkeit, den Prozessor wieder anzuhalten, um sein Programm zu ändern, falls dies gewünscht wird.

Die Ankopplung an die Umsetzersubsysteme erfolgt über FIFO-Speicher. Dafür gibt es verschiedene Gründe: Da der Prozessor TMS320 einen sehr kurzen Buszyklus hat, empfiehlt sich eine Trennung der zeitkritischen Prozessorseite von den zeitunkritischeren Umsetzersubsystemen. Ein weiterer Grund liegt in der begrenzten Anzahl von Schnittstellenadressen. Der TMS320 bietet selbst nur acht Schnittstellenadressen. Bei diesem Beispiel wurde das Problem so gelöst, daß der FIFO-Speicher für die ADU-Subsysteme unter einer Schnittstellenadresse anzusprechen ist und der FIFO-Speicher für die DAU-Subsysteme unter einer anderen Adresse. Da die FIFOs eine Speichertiefe von 16 Bit haben, können über die zeitliche Reihenfolge bis zu 16 ADU- bzw. DAU-Subsysteme bedient werden (*Bild 2*). Bezogen auf die ADU-Subsysteme heißt dies, daß als erstes der Digitalwert vom ADU 1 in den FIFO-Speicher geschrieben wird und danach der Digitalwert vom ADU 2 usw. Sind die Werte aller ADU-Subsysteme im FIFO-Speicher, so wird dies dem Prozessor über eine Statusleitung mitgeteilt. Hat der Prozessor Daten für die DAU-Subsysteme, schreibt er diese in den entsprechenden FIFO-Speicher. Danach werden die Werte, durch die Steuerkarte gelenkt, auf die einzelnen DAUs verteilt.

2.2 Umsetzer-Subsysteme

Das Analog/Digital-Umsetzer-System enthält einen 12-Bit-Umsetzer mit einer Umsetzzeit von 25 µs. Dieser Wert ist nicht übermäßig kurz, fällt aber nur einmal pro Abtastschritt an, da die Umsetzer parallel angeordnet sind. Diese Zeit kann bei umfangreicheren Regleralgorithmen als Rechenzeit genutzt werden, so daß die Abtastzeit durch die Umsetzzeit nicht erhöht wird.

Mit Hilfe des zentralen Abtasttakts wird die Umsetzung gestartet. Das Ende der Umsetzung wird durch eine Pegeländerung auf der Statusleitung angezeigt. Dieses Signal veranlaßt, daß die Daten vom Umsetzer in D-Flipflops zwischengespeichert werden. Das Statussignal wird auch zur Steuerkarte geführt. Dort werden die Statussignale aller angeschlossenen ADUs miteinander verknüpft, um den Zeitpunkt festzustellen, wann alle Umsetzvorgänge beendet sind, denn erst danach wird mit der Datenübertragung von den ADU-Systemen zum FIFO-Speicher auf der Prozessorkarte begonnen. Einzelne Umsetzersysteme werden durch eine 4-Bit-Adresse angesprochen. Soll eine Datenübertragung vom Umsetzersystem zum FIFO-Speicher stattfinden, sendet die Steuerkarte die jeweilige Adresse. Das angesprochene Umsetzersystem hebt den Tristate-Zustand der D-Flipflops auf, wodurch die Daten in den FIFO-Speicher übernommen werden können.

Als Digital/Analog-Umsetzer wird ein Baustein verwendet, der zwei 12-Bit-Registerblöcke enthält, die hintereinander geschaltet sind, aber zeitlich getrennt geladen werden können. Dadurch ist es möglich, neue Daten in den Umsetzer zu übernehmen, ohne die momentanen Werte im zweiten Registerblock, die umgesetzt werden, zu ändern. Erst durch ein zusätzliches Steuersignal wer-

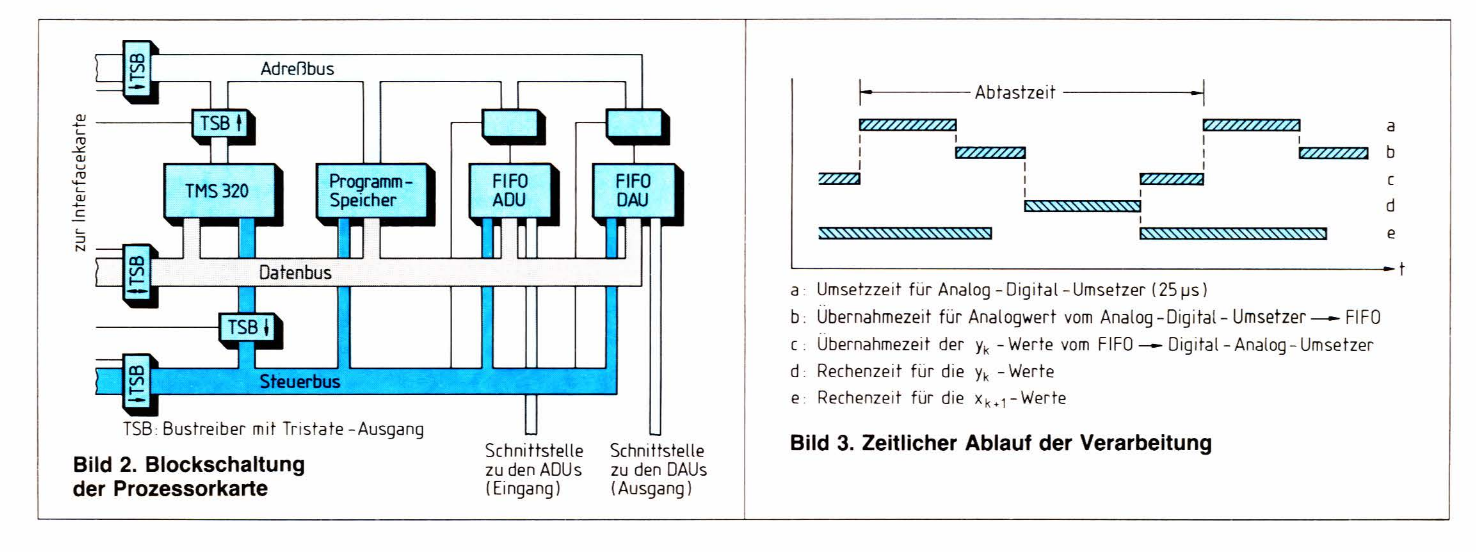


Bild 2. Blockschaltung der Prozessorkarte

Bild 3. Zeitlicher Ablauf der Verarbeitung

den diese Daten in den zweiten Registerblock übernommen und am Ausgang wirksam. Dies ist dann von Vorteil, wenn im Fall mehrerer paralleler DAU-Systeme die Werte gleichzeitig ausgegeben werden sollen, aber nur zu verschiedenen Zeiten an die einzelnen Systeme übergeben werden können. Die Umsetzersysteme werden über eine 4-Bit-Adresse angesprochen. Soll eine Datenübertragung vom FIFO-Speicher zum DAU-System stattfinden, sendet die Steuerkarte die jeweilige Adresse. Das angesprochene System kann dann auf die notwendigen Steuersignale zum Übernehmen der Daten reagieren.

2.3 Steuerkarte

Auf dieser Karte befindet sich die Steuerung der Datenübertragung zwischen den FIFO-Speichern auf der Prozessorkarte und den Umsetzersubsystemen. Sie generiert die notwendigen Steuersignale für die FIFO-Bausteine sowie die Adressen und Steuersignale für die Umsetzersubsysteme. Weiterhin wertet sie die Statussignale der angeschlossenen Analog/Digital-Umsetzer aus, um festzustellen, wann alle Umsetzvorgänge abgeschlossen sind. Außerdem wird der Abtasttakt für das Gesamtsystem von der Steuerkarte zur Verfügung gestellt.

2.4 Interfacekarte

Die Interfacekarte verbindet den digitalen Regelungsrechner mit einem Gastrechner. Mit deren Hilfe hat der Gastrechner die Möglichkeit, auf den Programmspeicher des TMS320 und die FIFO-Bausteine zuzugreifen. Über die FIFO-Bausteine hat der Gastrechner indirekt Zugriff auf die Umsetzsubsysteme. Das Reglerprogramm wird auf dem Gastrechner erstellt und dann über die Interfacekarte in den Programmspeicher des TMS320 übertragen. Danach werden die Leitungen der Interfacekarte in den hochohmigen und die Leitungen des TMS320 in den aktiven Zustand geschaltet. Damit läuft der Prozessor selbständig mit dem vorher eingespeicherten Programm. Bei notwendigen Änderungen kann man jederzeit eingreifen, um z. B. ein neues Programm zu laden. Durch Austauschen der Interfacekarte lassen sich beliebige Gastrechner anschließen.

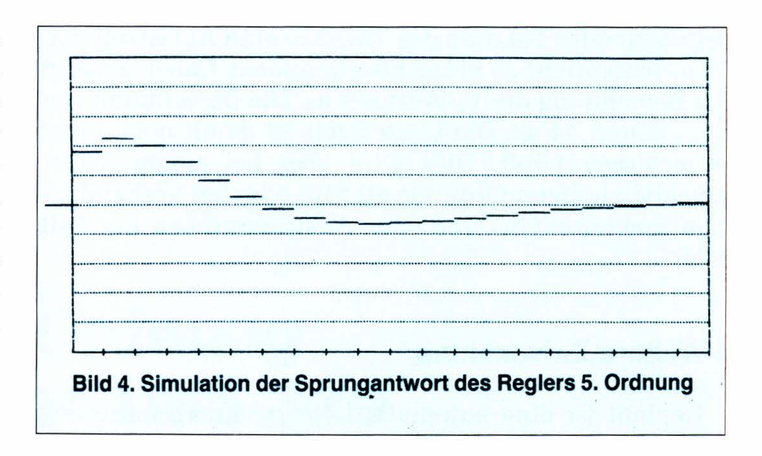


Bild 4. Simulation der Sprungantwort des Reglers 5. Ordnung

3 Programmrealisierung

Aufgebaut wurde ein Regler 5. Ordnung mit einem Eingang (Meßgröße u_k) und einem Ausgang (Steuergröße y_k). Dieser Regler war ursprünglich auf der Grundlage eines 16-Bit-µPs aufgebaut und diente einer industriellen Anwendung. Mit diesem Konzept wurde eine Abtastzeit von einigen Millisekunden erreicht.

Für die Realisierung findet der folgende Algorithmus [8] Verwendung:

$$\underline{x}_{k+1} = \underline{A}\ \underline{x}_k + \underline{B}\ u_k$$

$$y_k = \underline{C}\ \underline{x}_k + D\ u_k$$

Sobald die Eingangsgröße (Meßgröße u_k) umgesetzt vorliegt, wird y_k berechnet. Der dazu benötigte Wert $\underline{x}_k$ liegt vom vorhergehenden Abtastschritt vor. Nach erfolgter Berechnung von y_k wird dieses an das DAU-Subsystem übergeben. Hiernach wird $\underline{x}_{k+1}$ berechnet, das dann für den nächsten Abtastschritt benötigt wird. Bei Verwendung des oben angegebenen Algorithmus läßt sich die Umsetz- plus Übergabezeit der Eingangsgröße u_k für die Berechnung des $\underline{x}_{k+1}$ nutzen (*Bild 3*). Bei Reglern höherer Ordnung mit vielen Arithmetikoperationen ist in der Regel die Zeit für die Berechnung von $\underline{x}_{k+1}$ größer als die Zeit für die Umsetzung und Übergabe der Meßgrößen, so daß letztere die Abtastzeit nicht erhöhen. Dadurch können auch relativ langsame Umsetzer Verwendung finden. Für den realisierten Regler 5. Ordnung ergibt sich

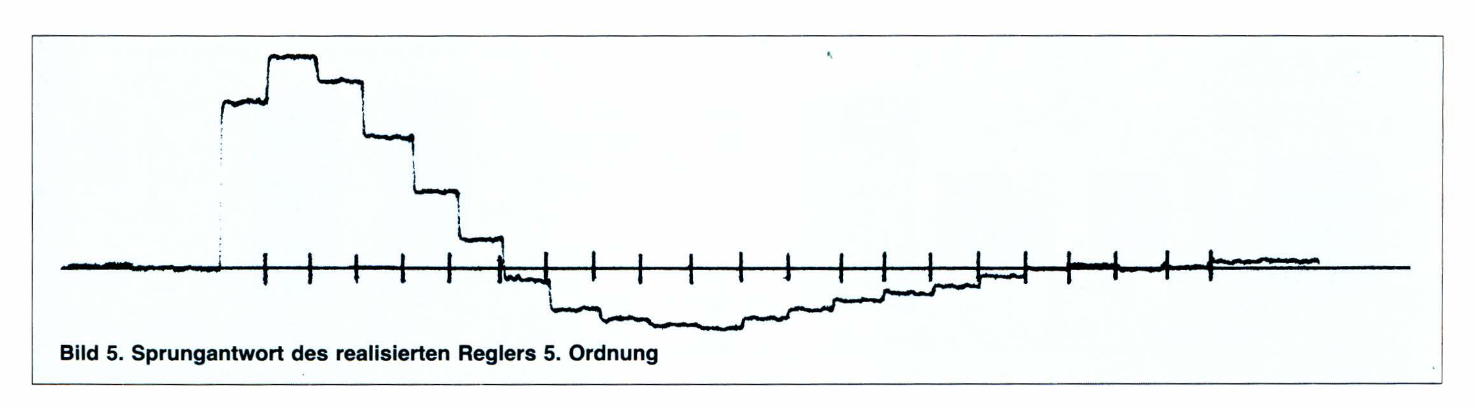


Bild 5. Sprungantwort des realisierten Reglers 5. Ordnung

bei maximaler Taktrate des TMS320 eine Abtastzeit von 50 µs (entspricht 20 kHz Abtastfrequenz). Dabei benötigt die Berechnung des y_k-Wertes 9 µs. Die Berechnung von $\underline{x}_{k+1}$ dauert 14 µs. Die Umsetzzeit ist damit noch nicht ganz ausgeschöpft. Dies wird aber bei einem Regler erheblich höherer Ordnung als fünf oder bei Vorhandensein mehrerer Ein- und/oder Ausgangsgrößen der Fall sein.

4 Weitere Entwicklung

Geplant ist eine automatische Programmgenerierung für Mehrgrößen-Regler auf einem Gastrechner (ähnlich wie in [8] beschrieben). Der Benutzer soll nur noch die Reglerdaten (Reglerkoeffizienten) in den Rechner eingeben. Der Programmgenerator erstellt auf dem Gastrechner ein lauffähiges Programm für den TMS320. Hierdurch wird der Benutzer von der schwierigen Programmentwicklung entlastet. Um das erstellte Programm vorher zu kontrollieren, soll noch ein Simulationsprogramm erstellt werden, mit dem der Prozessor TMS320 auf dem Gastrechner simuliert wird. Das Ergebnis der Simulation (z. B. Sprungantwort) wird dann grafisch auf dem Bildschirm des Gastrechners dargestellt.

Literatur

[1] *Ackermann, J.*: Abtastregelung. Berlin, Springer 1972.
[2] *Isermann, R.*: Digitale Regelsysteme. Springer, 1977.
[3] *Kolb, H.-J.*: Prozessorkonzepte zur digitalen Signalverarbeitung. ELEKTRONIK 1982, H. 21, S. 107...114.
[4] *Schloß, J., Müller, H.*: Schneller Vektorprozessor zum Anschluß an 16-Bit-Mikrocomputer. ELEKTRONIK 1982, H. 21, S. 100...104.
[5] *von Bechen, P.*: 32-Bit-Mikrocomputer für die Signalverarbeitung und Prozeßsteuerung. ELEKTRONIK 1982, H. 22, S. 139...141.
[6] *Mc Donouyh, K., Caudel, E., Magar, S., Leigh, A.*: Microcomputer with 32-bit arithmetic does high-precision numer crunching. Electronics, February 24, 1982, S. 105...110.
[7] TMS320 Preliminary Functional Specification. Texas Instruments.
[8] *Hanselmann, H.*: Tischrechner programmiert Signalprozessor als digitalen Mehrgrößenregler. ELEKTRONIK 1982, H. 21, S. 134...138.

Dipl.-Ing. Wolfgang Loges

Regelsysteme höherer Ordnung mit dem Signalprozessor TMS 320

Ein System mit dem Signalprozessor TMS 320 stößt bei Regelungs- und Filtersystemen hoher Ordnung auf Grenzen, die in der geringen Speicherkapazität des Daten-RAMs begründet sind. Gemeint ist damit, daß die Koeffizienten, die als 16-Bit-Zweierkomplement-Zahlen angenommen werden, nicht mehr geschlossen in das Daten-RAM passen. Der folgende Beitrag zeigt Möglichkeiten, wie auch in diesen Fällen mit dem Prozessor TMS 320 brauchbare Lösungen zu erzielen sind.

In der digitalen Regeltechnik ist Skalarproduktbildung die zentrale Rechenoperation. Der für die Skalarproduktbildung notwendige Multiplikationsbefehl des TMS 320 – für 16-Bit-Festkomma-Werte – setzt voraus, daß die Zahlen im Daten-RAM stehen. Bei Mehrgrößen-Regler hoher Ordnung mit mehreren Meßeingängen und mehreren Steuerausgängen kann die Koeffizientenzahl so groß werden, daß sie nicht mehr geschlossen im Daten-RAM unterzubringen ist. Um den TMS 320 auch bei diesen Anwendungen nutzen zu können, bedarf es besonderer Maßnahmen. Dieser Aufsatz beschreibt drei Verfahren, die eine Nutzung des TMS 320 in diesen Fällen ermöglichen. Um die drei Verfahren miteinander vergleichen zu können, wird nicht von dem üblichen Algorithmus für Mehrgrößen-Regler

$$\underline{x}_{k+1} = \underline{A}\,\underline{x}_k + \underline{B}\,\underline{u}_k$$
$$\underline{y}_k = \underline{C}\,\underline{x}_k + \underline{D}\,\underline{u}_k$$

ausgegangen, sondern von der folgenden Form der Berechnung eines Vektors:

$$\underline{e} = \underline{F}\;\underline{h}$$

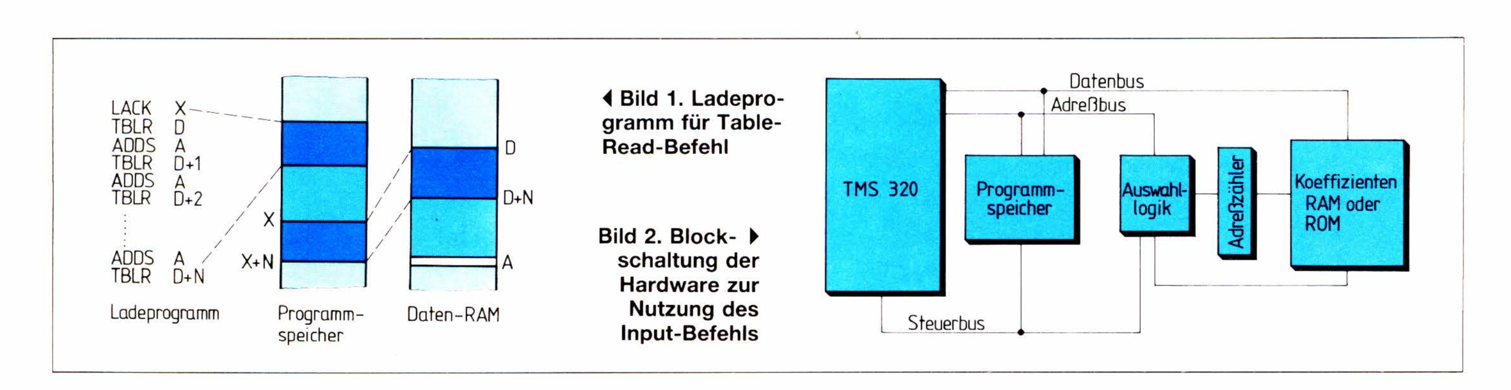
$\underline{F}$ ist eine $n \times n$-Matrix.

Bei Verwendung dieses Algorithmus liegt die Grenze dafür, daß alle Werte im Daten-RAM gespeichert werden können, bei $n = 11$. Dabei ergeben sich für die $\underline{F}$-Matrix 121 Elemente und für die Vektoren $\underline{e}$ und $\underline{h}$ jeweils 11 Elemente, so daß insgesamt 143 Elemente abgespeichert werden müssen.

1 Die Verfahren

1.1 *Anwendung des „Table-Read"-Befehls (TBLR)*

Der „Table-Read"-Befehl bietet die Möglichkeit, Daten, die im Programmspeicher stehen, in das Daten-RAM zu laden. Dazu muß vorher die Programmspeicheradresse im Akkumulator vorhanden sein. Bei Ausführung des Table-Read-Befehls wird der Inhalt des Akkumulators in den Programmzähler geladen. Sollen mehrere Daten vom Programmspeicher in das Daten-RAM übertragen werden, ist es vorteilhaft, die Daten hintereinander im Programmspeicher abzulegen. Das Programm für den Datentransfer sollte so gestaltet sein, daß die Adresse, unter der der Wert im Programmspeicher zu finden ist, mit dem Befehl „LACK" in den Akkumulator geladen wird (*Bild 1*). Durch den nachfolgenden Table-Read-Befehl wird der erste Wert ins Daten-RAM



◀ **Bild 1. Ladeprogramm für Table-Read-Befehl**

Bild 2. Blockschaltung der Hardware zur Nutzung des Input-Befehls ▶

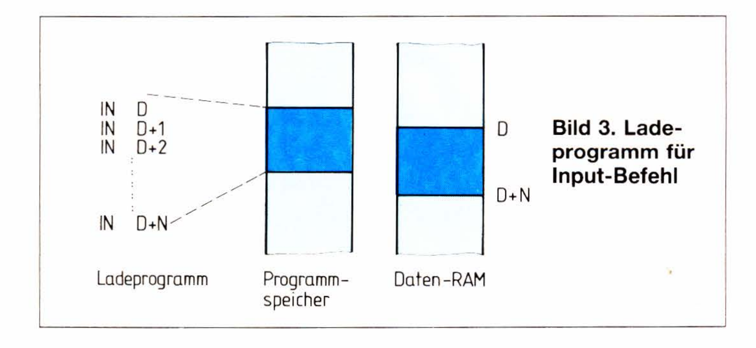


Bild 3. Ladeprogramm für Input-Befehl

Tabelle 1. Aufspaltung der Werte

Positive Werte	Negative Werte
	1000000000000000
0001000000000000	1001000000000000
0010000000000000	1010000000000000
0011000000000000	1011000000000000
0100000000000000	1100000000000000
0101000000000000	1101000000000000
0110000000000000	1110000000000000
0111000000000000	1111000000000000

geladen. An welche Stelle im Daten-RAM der Wert gespeichert wird, steht bei direkter Adressierung im Table-Read-Befehl. Durch Erhöhen des Akkumulatorinhalts um eins – über Additionsbefehl – kann mit dem Table-Read-Befehl der Inhalt der nächsten Programmspeicherstelle ins Daten-RAM übertragen werden. Bietet der Programmspeicher genügend Platz, sollte das Programm nicht als Schleife geschrieben sein, sondern für jedes zu übertragende Datenwort aus der Befehlskombination TBLR/ADDS bestehen, um hier nicht unnötig Zeit zu verschenken. Die benötigte Zeit für die Übertragung eines Datenwortes beträgt 800 ns, die sich aus 600 ns für den Table-Read- und 200 ns für den Additionsbefehl (ADDS) zusammensetzen.

1.2 Anwendung des „Input"-Befehls (IN)

Mit dem Input-Befehl können von insgesamt acht adressierbaren Ports 16-Bit-Daten in das Daten-RAM geladen werden. Soll dieser Befehl genutzt werden, bedarf es zusätzlicher Hardware. Sie besteht aus einem RAM oder ROM, in dem die Koeffizienten gespeichert sind, einem Adreßzähler, der nach jedem Datenzugriff durch den Input-Befehl um eins erhöht wird, und der notwendigen Auswahllogik, damit der Speicher durch die Portadresse angesprochen werden kann (*Bild 2*). Das notwendige Ladeprogramm kann, wenn der Programmspeicher genügend Platz bietet, aus der Hintereinanderreihung von Input-Befehlen bestehen (*Bild 3*). Dadurch liegt man zeitgünstiger als bei einer Schleifenprogrammierung. Benutzt wird bei dem Ladeprogramm direkte Adressierung, so daß im Input-Befehl mit angegeben wird, an welche Stelle im Daten-RAM die Werte abgelegt werden sollen. Das Laden eines Datenwortes benötigt hier eine Zeit von 400 ns.

```
ZAC
LT    A
MPYK
LTA   A
MPY   A
LTA   A
MPYK
:
:
:
:
:
LTA   A
MPYK
LTA   A
MPY   A
APAC
SACH  A
```

◀ Bild 4. Programm zur Skalarproduktberechnung mit dem Befehl „Multiply Immediate" (MPYK) (A: symbolische Adresse)

```
ZAC
LT    A
MPY   A
LTA   A
MPY   A
:
:
:
:
:
LTA   A
MPY   A
APAC
SACH  A
```

Bild 5. ▶ Programm zur Skalarproduktberechnung mit dem Befehl „Multiply" (A: symbolische Adresse)

A symbolisch

1.3 Anwendung des Befehls „Multiply Immediate" (MPYK)

Durch diesen Befehl wird eine Zahl im T-Register des Multiplizierers mit einer 13-Bit-Konstanten (Zweierkomplement) multipliziert. Der Wert der Konstanten ist im Multiplikationsbefehl MPYK enthalten. Würde die $\underline{F}$-Matrix nur 13-Bit-Koeffizienten enthalten, könnte man mit diesem Multiplikationsbefehl die Skalarprodukte direkt berechnen. Die Elemente des $\underline{e}$- und $\underline{h}$-Vektors müssen im Daten-RAM gespeichert sein, während die Koeffizienten der $\underline{F}$-Matrix in der Software enthalten sind. Die Überlegung ist nun, wie dieser Multiplikationsbefehl MPYK auch bei 16-Bit-Koeffizienten zu nutzen ist. Diese Möglichkeit bietet sich, wenn die 16-Bit-Koeffizienten f_{ui} der $\underline{F}$-Matrix in einen 13-Bit-Anteil f_{ui}' und einen 16-Bit-Anteil f_{ui}'' aufgespalten werden, so daß gilt: $f_{ui} = f_{ui}' + f_{ui}''$. Der 13-Bit-Anteil ist – wie oben erwähnt – im Multiplikationsbefehl MPYK enthalten. Der 16-Bit-Anteil muß im Daten-RAM abgespeichert werden. Der Vorteil dieser Aufspaltung liegt darin, daß für den 16-Bit-Anteil insgesamt nur 15 verschiedene Zahlen auftreten (siehe *Tabelle 1*). Wendet man diese Aufspaltung bei der Skalarproduktbildung an, lassen sich die Terme

$$e_u = \sum_{i=1}^{n} f_{ui} h_i \quad \text{in}$$

$$e_u = \sum_{i=1}^{n} (f_{ui}' + f_{ui}'') h_i$$

$$= \sum_{i=1}^{n} (f_{ui}' h_i + f_{ui}'' h_i)$$

umwandeln.

Das Programm für den TMS 320 zur Berechnung der Skalarprodukte unter Anwendung der oben angegebe-

nen Aufspaltung der 16-Bit-Koeffizienten f_{ui} zeigt *Bild 4*. Für ein Skalarprodukt ergibt sich für das Programm nach Bild 4 eine Rechenzeit von t_1 = n 800 ns + 600 ns, wenn der TMS 320 mit 20-MHz-Takt läuft. Bei der Anwendung dieses Verfahrens werden im Daten-RAM 15 Speicherplätze für die 16-Bit-Anteile f_{ui}'' (Tabelle 1) und jeweils n Speicherplätze für die Vektoren $\underline{e}$ und $\underline{h}$ belegt. Von den 144 vorhandenen Speicherplätzen wird somit nur ein Teil benutzt.

Die noch freien Speicherplätze sollten ebenfalls zu nutzen sein. Als Möglichkeit bietet sich hier an, einen Teil der 16-Bit-Koeffizienten f_{ui} der $\underline{F}$-Matrix direkt ins Daten-RAM zu speichern. Die Skalarprodukte mit diesen Koeffizienten werden mit dem Programm berechnet, das in *Bild 5* dargestellt ist. Durch Ausnutzen dieser Möglichkeit verringert sich die Rechenzeit für die Berechnung des $\underline{e}$-Vektors gegenüber der alleinigen Anwendung des Programms nach Bild 4. Als Rechenzeit ergibt sich bei insgesamt n Skalarprodukten t_2 = (m 400 ns + 600 ns) + ([n − m] 800 ns + 600 ns), wenn angenommen wird, daß m Skalarprodukte mit dem Programm nach Bild 5 berechnet werden.

2 Zeitbedarf für die Berechnung des Vektors $\underline{e}$

In *Tabelle 2* ist für die drei Verfahren die Rechenzeit für den Vektor $\underline{e}$ für verschiedene n der $\underline{F}$-Matrix zusammengestellt. Dabei wird von n > 11 ausgegangen, da die Koeffizienten bis n = 11 sich noch geschlossen im Daten-RAM speichern lassen. Das Programm für die Berechnung der Skalarprodukte für das Verfahren „Table Read" und „Input" ist in Bild 5 zu sehen. Als Rechenzeit ergibt sich bei den beiden Verfahren für ein Skalarprodukt t_3 = n 400 ns + 600 ns. Für das Verfahren „Multiply Immediate" wird die Rechenzeit t_2 zugrunde gelegt, wie in 1.3 angegeben. In den Tabellen 2a und 2b ist neben der Zeit für die Berechnung des Vektors $\underline{e}$ auch die Ladezeit mit aufgeführt. In *Bild 6* sind für die drei Verfahren die benötigten Zeiten für die Berechnung des Vektors $\underline{e}$ in Abhängigkeit von n grafisch dargestellt. Ab n = 15 ist die Koeffizientenzahl von $\underline{F}$ so groß, daß bei den beiden Verfahren „Table Read" und „Input" während der Berechnung des Vektors $\underline{e}$ mehrmals umgeladen werden muß.

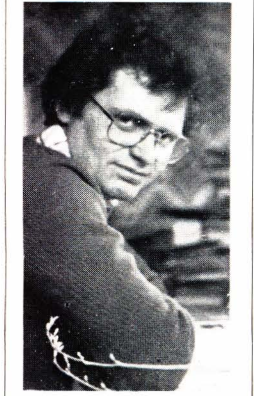
Der Beitrag sollte eine Anregung geben, wie der TMS 320 auch in Systemen eingesetzt werden kann, bei denen die Koeffizientenzahl größer ist als der Speicherplatz im Daten-RAM. Es ergibt sich, daß der Table-Read-Befehl nicht immer die zeitoptimale Lösung bietet. Günstiger liegt man mit dem Input-Befehl, was aber mit zusätzlicher Hardware erkauft werden muß. Bei einer Koeffizientenzahl größer 200 bietet das in 1.3 vorgestellte Verfahren die zeitlich bessere Lösung, weil bei den beiden anderen Verfahren hier mehrmals während der Berechnung des Vektors $\underline{e}$ umgeladen werden muß.

Literatur

[1] TMS 320 Preliminary Functional Specification. Texas Instruments.

[2] *v. Bechen, P.*: 32-Bit-Mikrocomputer für die Signalverarbeitung und Prozeßsteuerung. ELEKTRONIK 1982, H. 22, S. 139...141.

[3] *Loges, W.*: Ein schnelles Signalprozessorsystem mit dem TMS 320 für die digitale Regelung. ELEKTRONIK 1983, H. 19, S. 51...54.



Dipl.-Ing. Wolfgang Loges wurde in Paderborn geboren. Er studierte an der Ruhr-Universität Bochum Elektrotechnik. Seit 1979 ist er wissenschaftlicher Mitarbeiter in der Fachgruppe Automatisierungstechnik (Prof. Dr.-Ing. J. Lückel) an der Universität-Gesamthochschule Paderborn. Neben seinem Hauptaufgabengebiet der Realisierung schneller digitaler Mehrgrößenregler widmet er sich noch der elektrischen Meßtechnik.

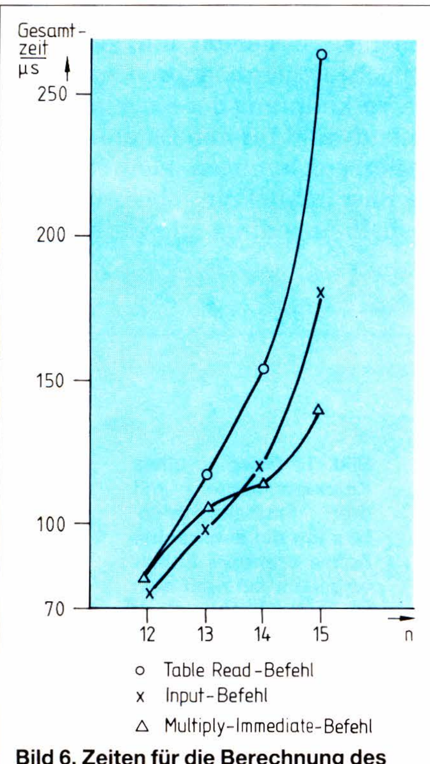


Bild 6. Zeiten für die Berechnung des Vektors $\underline{e}$ in Abhängigkeit von n

Tabelle 2. Rechenzeiten für den Vektor $\underline{e}$

Tabelle 2a

n	Koeffizienten-Zahl für $\underline{F}$	t_3 µs	Ladezeit µs	Gesamtzeit µs
12	144	64,8	19,2	84
13	169	75,4	41,6	117
14	196	86,8	67,2	154
15	225	99,0	164,0	263

Tabelle 2b

n	Koeffizienten-Zahl für $\underline{F}$	t_3 µs	Ladezeit µs	Gesamtzeit µs
12	144	64,8	9,6	74,4
13	169	75,4	20,8	99,2
14	196	86,8	33,6	120,4
15	225	99,0	82,0	181,0

Tabelle 2c

n	Koeffizienten-Zahl für $\underline{F}$	t_3 µs	t_1 µs	m	(Gesamtzeit) t_2 µs
12	144	43,2	40,8	8	84,0
13	169	40,6	66,0	7	106,6
14	196	43,4	70,8	7	114,2
15	225	39,6	100,8	6	140,4

Dipl.-Ing. Peter Rojek, Walter Wetzel

Multiprozessorkonzept für Industrieroboter:

Mehrgrößenregelung mit Signalprozessoren

Universell einsetzbare Industrieroboter verfügen über sechs oder mehr Freiheitsgrade mit entsprechend vielen – meist elektrischen – Antriebseinheiten. Die Drehzahl- und Stromregelung der Maschinen erfolgt üblicherweise mit festeingestellten Analogreglern, während die weniger zeitkritische, aber genaue Lageregelung digital ausgeführt ist. Für eine vollständige digitale Regelung ist die komplexe Regelstruktur eines Roboters in Teilaufgaben zu gliedern, die gleichzeitig von Komponenten eines Multiprozessorsystems bearbeitet werden können. Im folgenden wird ein Rechnerkonzept vorgestellt, bei dem die Kopplung zwischen den Prozessoren im Hinblick auf die speziellen Echtzeitanforderungen optimiert ist. Die Rechenleistung steigt durch den Einsatz von Signalprozessoren (TMS 320) beträchtlich, so daß eine hohe Freizügigkeit hinsichtlich der auszuführenden Regelalgorithmen erreichbar ist.

Moderne Signalprozessoren erlangen für die Regelungstechnik wachsende Bedeutung, denn sie ermöglichen bisher vorzugsweise theoretisch erprobte, arithmetisch aufwendige Regelverfahren auch praktisch einzusetzen. Als Beispiel, das für eine antriebstechnische Weiterentwicklung bei Industrierobotern von Bedeutung ist, sei hier die digitale Regelung kontaktfreier Drehstromstellantriebe genannt. Der Einsatz eines Signalprozessors ist erforderlich, um mit möglichst kurzen Abtastintervallen eine hohe Bandbreite zu erreichen [1, 9]. Drehstrommotoren sind wartungsfrei, stärker belastbar und in Verbindung mit Transistorstellgliedern hoher Schaltfrequenz von besserer Dynamik als bislang bei Robotern eingesetzte Gleichstrommotoren.

Werden Elemente einer Antriebsregelung durch Software ersetzt, so sinkt der Aufwand für Bauteile; Driftprobleme werden umgangen, und Reglereinstellungen können in definierter Weise im Rechner erfolgen. Mit einer digitalen Messung der Winkelstellung läßt sich die Drehzahl durch Differentiation leicht ermitteln, so daß ein Tachogenerator nicht erforderlich ist.

Für hochdynamische Stellantriebe bietet die Kombination eines universellen Mikrorechners für die Ein-/Ausgabe mit einem Signalprozessor als Recheneinheit besonders günstige Voraussetzungen. Zur Regelung eines Roboters ist somit ein Multiprozessorsystem zu entwerfen, bei dem für jeden Antrieb ein eigener Rechner zur Verfügung steht. Zusätzliche Aufgaben ergeben sich durch die Verkopplung der einzelnen Achsen, die mittels geeigneter Entkopplungen des Reglers zu berücksichtigen sind.

Ein freizügig verwendbares Multiprozessorsystem sollte aus standardisierten, modular gegliederten Komponenten bestehen, um jederzeit Erweiterungen und einfache Fehlerdiagnosen zu erlauben. Problematisch ist die Verknüpfung der einzelnen Recheneinheiten, da der Zeitaufwand für den Datentransfer im Vergleich zu den Reglerberechnungen klein bleiben muß. Eine auf die Rechnerarchitektur abgestimmte Koppelsoftware ist deshalb unbedingt erforderlich.

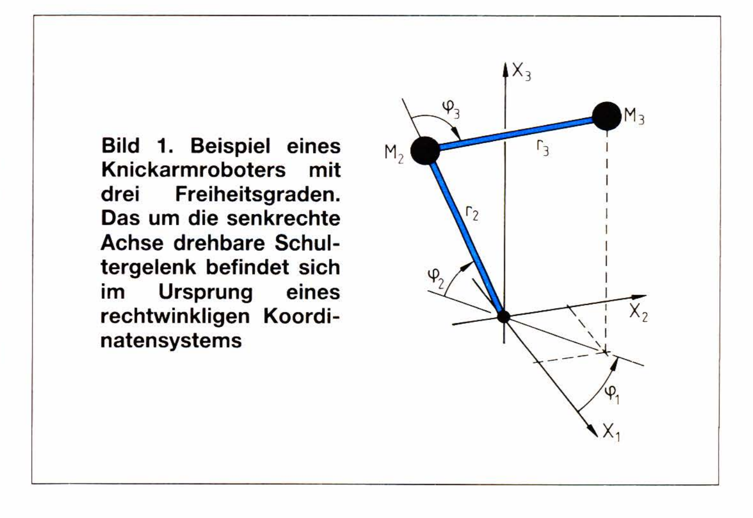


Bild 1. Beispiel eines Knickarmroboters mit drei Freiheitsgraden. Das um die senkrechte Achse drehbare Schultergelenk befindet sich im Ursprung eines rechtwinkligen Koordinatensystems

Modell mit drei Freiheitsgraden

Am Beispiel eines Gelenkroboters mit drei Freiheitsgraden lassen sich die prinzipiellen Probleme einer digitalen Regelung und die Realisierung mit einem Multiprozessorsystem deutlich machen (*Bild 1*). Verstellungen werden durch drei den einzelnen Gelenken zugeordnete Antriebe vorgenommen. In den Gelenken und in der Hand sind konzentrierte Massen M_i vorausgesetzt, die Armteile seien starr und masselos.

Die Bewegungen eines solchen Robotermodells sind mit nichtlinearen, zeitvarianten Differentialgleichungen beschrieben [2]. Wegen der variablen Geometrie ändern sich die für die einzelnen Antriebe wirksamen Trägheitsmomente:

$$J_1 = M_2 \cdot (r_2 \cdot \cos\varphi_2)^2 + M_3 \cdot \left((r_2 \cdot \cos\varphi_2 + r_3 \cdot \cos)\,\varphi_2 + \varphi_3\right)^2$$

$$J_2 = M_2 \cdot r_2^2 + M_3 \cdot (r_2^2 + r_3^2 + 2 \cdot r_2 \cdot r_3 \cdot \cos\varphi_3)$$

$$J_3 = M_3 \cdot r_3^2$$

Das Gerät arbeitet einschließlich der Meßwerterfassung in gelenkbezogenen Koordinaten (φ_i). Will man aber Punkte in einem raumfesten, kartesischen Koordinatensystem vorgeben, so ist eine Koordinatentransformation durchzuführen, die wegen der trigonometrischen Funktionen einen erheblichen Rechenaufwand bedingt:

$$x_1 = -\left(r_2 \cdot \cos\varphi_2 + r_3 \cdot \cos(\varphi_2 + \varphi_3)\right) \cdot \cos\varphi_1$$

$$x_2 = -\left(r_2 \cdot \cos\varphi_2 + r_3 \cdot \cos(\varphi_2 + \varphi_3)\right) \cdot \sin\varphi_1$$

$$x_3 = r_2 \cdot \sin\varphi_2 + r_3 \cdot \sin(\varphi_2 + \varphi_3)$$

Um das Gerät in Raumkoordinaten (x_i) regeln zu können, muß man diese Gleichungen in wenigen Millisekunden nach den Gelenkkoordinaten (φ_i) auflösen.

Die Regelung dieses Modellroboters kann mit Hilfe dreier mehrschleifiger Kaskadenregelungen erfolgen (*Bild 2*), wobei je ein Lage-, ein Drehzahl- und ein Stromregelkreis ineinander verschachtelt sind. Den Rahmen bilden die Transformation vom raumfesten Koordinatensystem ins Gelenkkoordinatensystem und die Rücktransformation. Um die zwischen den einzelnen Achsen bestehenden statischen und kinematischen Verkopplungen auszugleichen, können am Ausgang der Drehzahlregler die Stromsollwerte innerhalb vorgegebener Begrenzung modifiziert werden, z. B. in Form einer multiplikativen Verknüpfung mit dem jeweiligen Trägheitsmoment. Diese Anpassung an die variablen Parameter der Regelstrecke erfordert wiederum einen erheblichen Rechenaufwand und kann nur digital erfolgen. Insgesamt entsteht dadurch ein komplexes Mehrgrößen-Regelsystem mit adaptiven Zusatzfunktionen.

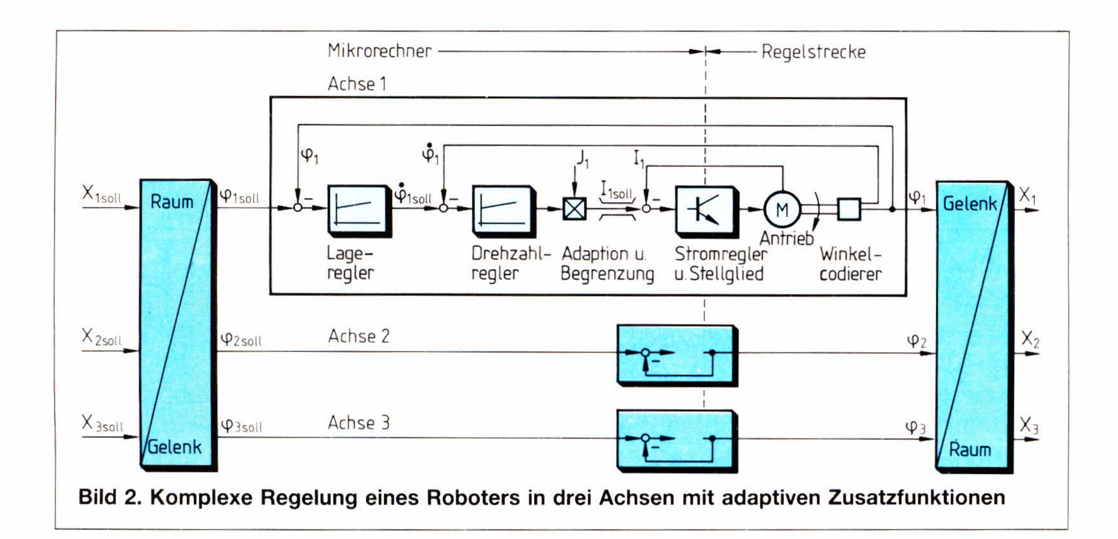


Bild 2. Komplexe Regelung eines Roboters in drei Achsen mit adaptiven Zusatzfunktionen

Doppelprozessorkarte mit Hauptprozessor...

Das im *Kasten* auf der gegenüberliegenden Seite gezeigte Multiprozessorsystem besteht aus mehreren baugleichen Doppelprozessorkarten, deren Kernstück ein Prozessor aus der 16-Bit-Generation TMS 99000 bildet. Diese Prozessoren (TMS 99105 bzw. TMS 99110) zeichnen sich durch eine Speicher-zu-Speicher-Architektur aus, die eine kurze Interrupt-Antwortzeit (2,2 µs bei 24-MHz-Takt) gewährleistet. Weitere wichtige Merkmale dieser Prozessoren sind:

- 167 ns Zykluszeit;
- 16 Interruptebenen;
- 256 KByte adressierbarer Speicher;
- 16 KBit Ein-/Ausgabe (Einzelbit);
- 16 KWorte Ein-/Ausgabe (16 Bit parallel);
- integrierter Taktoszillator und
- Floating-Point-Befehle (nur TMS 99110).

Da die Prozessoren TMS 99105 und TMS 99110 anschlußkompatibel sind, ist eine wahlweise Bestükkung der Prozessorkarte je nach Anforderung möglich. Zur vollen Ausnutzung der Prozessorleistung müssen schnelle RAMs eingesetzt werden, die einen Zugriff innerhalb 55 ns gestatten.

Bild 3 zeigt die Blockschaltung der im Doppel-Europaformat aufgebauten Doppelprozessorkarte. Der Prozessor TMS 99105 besitzt 4 KWorte RAM, die ohne Wartezyklen betrieben werden. Weiterhin sind vier Sockel mit maximal 16 KWorten Speicher (wahlweise EPROM oder RAM) als Programm- und/oder Datenspeicher ansprechbar. An Peripherie stehen auf der Karte zwei serielle Schnittstellen (TMS 9902) mit RS-232-Spezifikation sowie ein Interrupt-Controller – kombiniert mit Zähler und Ein-/Ausgabeport (TMS 9901) – zur Verfügung. Diese Bausteine werden vom Hauptprozessor im bitseriellen E/A-Bereich angesprochen.

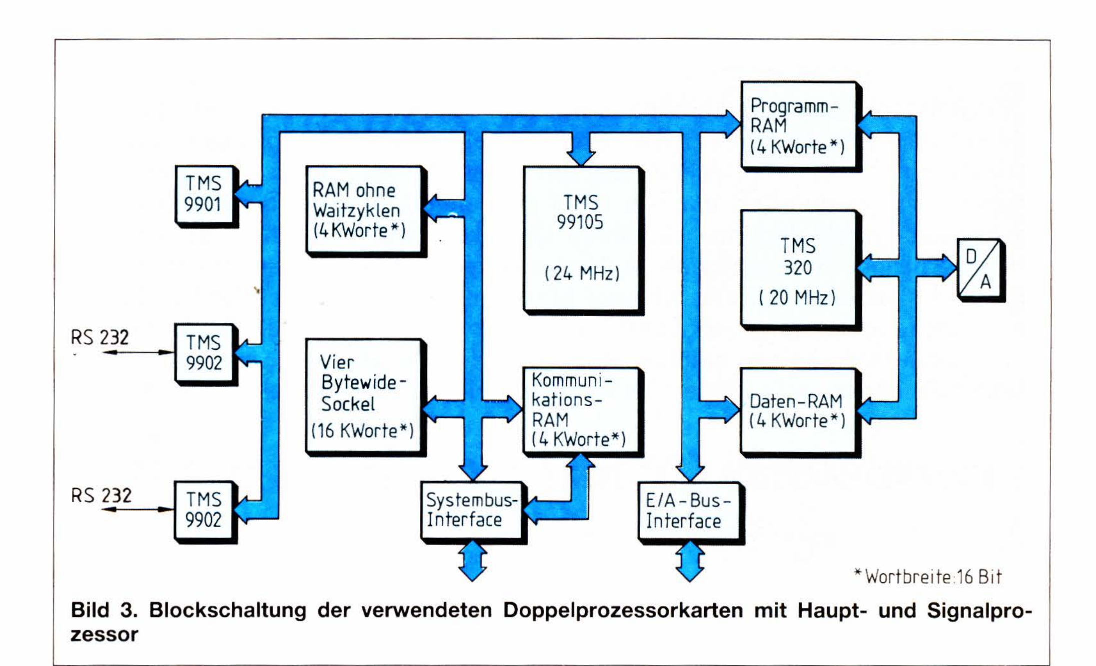


Bild 3. Blockschaltung der verwendeten Doppelprozessorkarten mit Haupt- und Signalprozessor

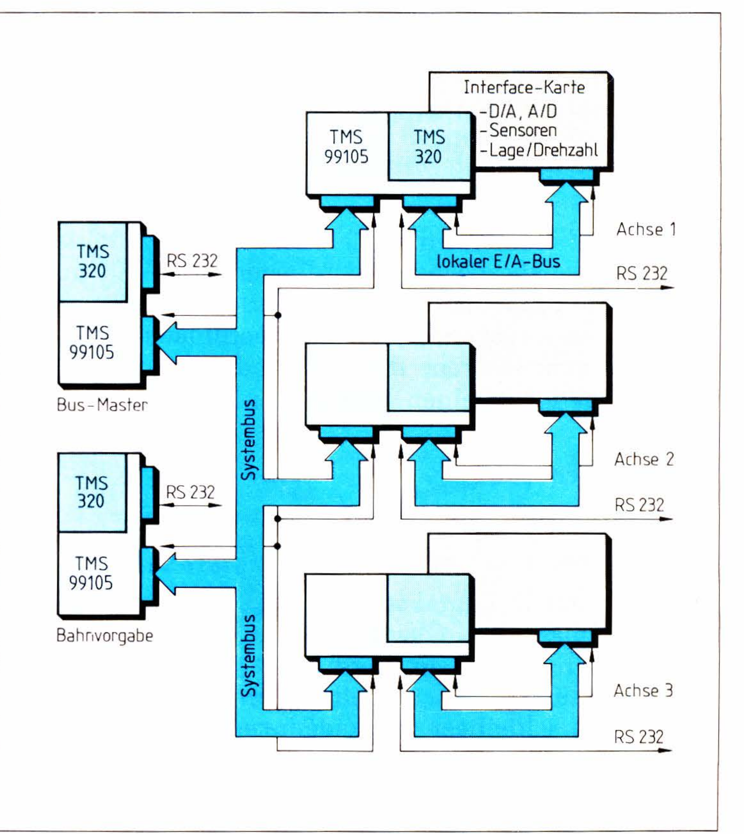
Über die beiden 96poligen Steckverbinder (DIN 41 612) der Baugruppe hat der TMS 99105 Zugriff zu zwei unterschiedlichen Bussystemen. Ein Businterface ist für den Aufbau eines lokalen Ein-/Ausgabebusses zum Anschluß der benötigten Sensoren (A/D-Umsetzer, Winkelcodierer usw.) sowie zur Ausgabe der ermittelten Sollwerte (D/A-Umsetzer) vorgesehen. Für diese Aufgaben stehen im parallelen 16-Bit-E/A-Bereich 8192 Adressen zur Verfügung. Weiterhin sind fast 16 K Ein-/Ausgabeadressen für die bitweise Adressierung von Peripherieeinheiten verwendbar. Zur schnellen Ankopplung der Peripherie sind acht der insgesamt 16 Interrupt-Ebenen des TMS 99105 vorgesehen.

Der zweite Steckverbinder der Doppelprozessorkarte dient zur Kopplung von maximal 17 gleichartigen Karten über einen gemeinsamen Systembus. Je nach Betriebsart der Baugruppe ist der Anschluß zum Systembus als aktives bzw. passives Interface konfiguriert. Der Datenaustausch zwischen den Doppelprozessorkarten erfolgt mit Hilfe eines 4 KWorte großen Kommunikationsspeichers und einer speziellen Kopplungssoftware.

Zur Synchronisation der über den Systembus gekoppelten Baugruppen sind fünf Interrupt-Eingänge auf den Bussteckverbinder geführt. Ein Power-Fail-Eingang, dessen Aktivierung einen nichtmaskierbaren Interrupt auslöst, ist für ein definiertes Abschalten des Gesamtsy-

Jedem Antrieb sein eigener Rechner

Die Struktur der Achsregelung spiegelt sich direkt in dem entwickelten Multiprozessorsystem wider: Für die Regelung jeder Roboterachse kommt eine Rechnerkarte, bestückt mit einem 16-Bit-Prozessor TMS 99000 und einem Signalprozessor TMS 320, zum Einsatz. Meßwerterfassung und Sollwertausgabe erfolgen über den jeweiligen lokalen E/A-Bus getrennt für jede Achse. Für übergeordnete Funktionen wie Bahnsteuerungen oder Bedienerkommunikation können weitere Doppelprozessorkarten eingefügt werden. Der Datenaustausch zwischen den einzelnen Modulen des Gesamtsystems ist über einen Bus-Master organisiert, der mit spezieller Koppelsoftware auszurüsten ist. Insgesamt ist ein Ausbau auf 16 selbständig arbeitende Recheneinheiten möglich, die miteinander über den Bus-Master korrespondieren. Damit ist gewährleistet, daß auch für andere Roboter mit zusätzlichen Freiheitsgraden ausreichend Rechenleistung zur Verfügung steht. Da es sich um ein Forschungsvorhaben handelt, steht gegenwärtig die funktionsmäßige Freizügigkeit im Vordergrund des Interesses; später muß natürlich eine Beschränkung des Aufwandes auf das unbedingt Notwendige erfolgen.



stems bei Spannungsausfall vorgesehen. Die Steckerbelegung des Systembusses erlaubt das Verwenden einer VME-Bus-Rückwand, deren spezifizierte Übertragungseigenschaften der hohen Datentransfergeschwindigkeit des eingesetzten Prozessors genügen.

...und Signalprozessor

Mit dem Signalprozessor TMS 320 (20-MHz-Takt) auf der Platine wird die für komplexe digitale Regelungen in Echtzeit benötigte Rechenleistung bereitgestellt. Durch eine interne 32-Bit-Architektur, einen parallelen 16×16-Bit-Multiplizierer und den 16 Bit breiten Datenbus erreicht der TMS 320 sehr kurze Ausführungszeiten. Zum Beispiel werden für eine 16×16-Bit-Multiplikation nur 200 ns benötigt.

Bild 4. Kopplung der Prozessorkarten mit Systembus-Schnittstelle für Bus-Master und -Slave

Auf der Doppelprozessorkarte stehen für den TMS 320 4 KWorte RAM als Programmspeicher zur Verfügung. Zusätzlich zu den internen 144 Worten Datenspeicher können extern weitere 4 KWorte RAM als Datenspeicher angesprochen werden. Da die Adressierfähigkeit des TMS 320 auf 4 KWorte Programm begrenzt ist, wird dieser Speicherbereich mit Hilfe eines Zählers adressiert. Damit ist die Möglichkeit gegeben, unabhängig vom Programmspeicher umfangreiche Koeffiziententabellen oder Regelungsparameter im Speicher bereitzustellen, die bei Bedarf in das TMS 320-interne RAM geladen werden.

Weiterhin läuft die Kommunikation des Signalprozessors mit dem Hauptprozessor TMS 99105 über den beschriebenen Speicherbereich. Vom TMS 320 direkt angesprochen werden zwei D/A-Umsetzer, die insbesondere zur Anzeige von berechneten Werten und zum Messen von Programmlaufzeiten vorgesehen sind.

Prozessorkopplung

Die CPU TMS 99105 überwacht Initialisierung und Funktion des Signalprozessors vollständig. Dazu gehört das Auslösen eines Interrupts per Software und die Möglichkeit, den TMS 320 in den Resetzustand zu versetzen bzw. zu aktivieren. Der gesamte Speicher (8 KWorte) des Signalprozessors liegt im direkten Adreßraum des TMS 99105 und ist somit von diesem unmittelbar beschreibbar.

Nach dem Rücksetzen des Signalprozessors vom TMS 99105 aus kann der Programmspeicher des TMS 320 mit dem auszuführenden Code geladen werden. Ebenso wird das Daten-RAM vom Hauptprozessor gesetzt. Mit der Rücknahme des Resetsignals am TMS 320 ist der Zugriff des TMS 99105 auf den Programmspeicher des Signalprozessors hardwaremäßig ausgeschlossen. Die Übergabe von Daten und Steuerbefehlen zwischen den beiden Prozessoren ist jetzt auf das 4 KWorte große Daten-RAM des Signalprozessors beschränkt.

Die Synchronisation von zwei auf den Prozessoren parallel ablaufenden Programmen geschieht über wechselseitige, per Software auszulösende Unterbrechungen und/oder die Abfrage von Semaphoren. Die gegenseitige Verriegelung der verschiedenen Prozessorbusse erfolgt durch Adreß- und Datenbusumschalter in Verbindung mit einer umfangreichen Steuerlogik. Bei einem gleichzeitigen Zugriff auf das als Koppelspeicher (Dual-Port-RAM) betriebene Daten-RAM wird der TMS 99105 durch Wartezyklen solange angehalten, bis der TMS 320 seinen Zugriff beendet hat. Abhängig vom momentan ausgeführten Opcode des Signalprozessors ergeben sich dadurch für den reinen Transfer eines 16-Bit-Datenwortes vom TMS 99105 zum TMS 320 und umgekehrt Zeiten zwischen 330 ns und maximal 670 ns.

Rechnerkopplung über Bus-Master und -Slaves

Ziel eines jeden Multiprozessorsystems auf der Basis von Mikroprozessoren ist es, ein Erhöhen des Gesamtdurchsatzes durch weitestgehende Parallelverarbeitung zu erreichen. Neben der Leistungsfähigkeit der eingesetzten Prozessoren sind die Architektur des Kommunikationssystems und die Organisation des Datenaustausches zwischen den Prozessoren ein entscheidender Punkt für die zu erreichende Systemleistung. Häufig dient zur Kopplung der einzelnen Baugruppen ein allen gemeinsamer Systembus (Multibus, VME-Bus usw.). Die von der Arbitrationslogik benötigte Zeit zur Zugriffszuteilung verringert allerdings die tatsächliche Datentransferrate erheblich. Für die vorliegende Problemstellung mit ihren Echtzeitanforderungen mußte daher eine andere Lösung gefunden werden.

Geht man von den speziellen Anforderungen einer digitalen Regelung für einen Roboter aus, so ergibt sich aus der Forderung nach einer möglichst hohen Abtastfrequenz eine entsprechende Datentransferrate. Dabei ist die Anzahl der pro Achsrechner in einem Abtastraster zu übertragenden Datenworte relativ gering (etwa 20...50). Allerdings werden die berechneten Werte eines Achsrechners gleichzeitig bei mehreren anderen Rechnern benötigt. Auf Grund dieser Situation wurde für den Datenaustausch zwischen den Systemeinheiten die im Kasten dargestellte Architektur verwirklicht.

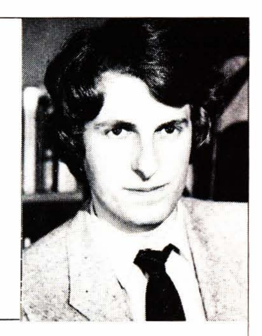
Der Austausch der Daten zwischen den Einheiten des Multiprozessorsystems wird von einer als Bus-Master (Steuerrechner) bezeichneten Prozessorkarte ausgeführt und erfolgt in einem festen Zeitraster, das dem Abtastraster entsprechen kann. Alle weiteren Baugruppen des Systems sind an diesem Vorgang nur passiv beteiligt, sie müssen lediglich die Transferdaten in einem eigens dafür vorgesehenen Kommunikationsspeicher bereitstellen. Der Ablauf eines Datentransfers ist dann wie folgt:

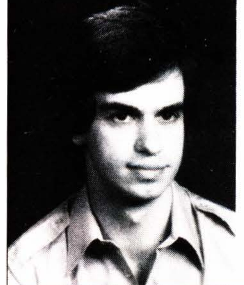
- Die zu übertragenden Daten werden vom Bus-Master nacheinander bei den Bus-Slaves (Teilrechner) eingesammelt. Dazu wird vom Bus-Master mittels einer Kartenadresse der jeweilige Bus-Slave selektiert und in den HOLD-Zustand versetzt. Nach dem Auslesen der Daten aus dem 4 KWorte großen Kommunikationsspeicher des Bus-Slaves wird vom Bus-Master das HOLD-Signal für diese Baugruppe wieder deaktiviert (bei den übrigen Bus-Slaves entsprechend).
- Nachdem die Daten vollständig eingesammelt sind, versetzt der Bus-Master alle angeschlossenen Bus-Slaves gemeinsam in den HOLD-Zustand. Dann werden die zuvor gesammelten Daten gleichzeitig in die Kommunikationsspeicher aller Bus-Slaves geschrieben.

Die hohe Transferrate des Systembusses (6 MByte/s) führt in Verbindung mit der Einsparung von Schreibzyklen zu einer wesentlichen Steigerung des Datendurchsatzes im Gesamtsystem.

Um gleiche Baugruppen zu erhalten, wird die Funktion des Bus-Masters bzw. eines Teilrechners von der gleichen Doppelprozessorkarte erfüllt. Die jeweilige Betriebsart der Baugruppe ist nur durch die zugehörige Software und das Umsetzen einiger Steckverbinder auf der Karte festgelegt. *Bild 4* zeigt die Systembus-Schnittstelle für den Bus-Master und für einen Bus-Slave.

Dipl.-Ing. Peter Rojek wurde in Bremen geboren und studierte an der TU Braunschweig Elektrotechnik mit dem Schwerpunkt Regelungstechnik. Seit 1983 ist er wissenschaftlicher Mitarbeiter am Institut für Regelungstechnik der TU Braunschweig; Arbeitsgebiet: Echtzeitsignalverarbeitung bei mehrachsigen Handhabungsgeräten (Industrierobotern).





Walter Wetzel wurde in Spangenberg (Nordhessen) geboren. Er studiert an der TU Braunschweig Elektrotechnik mit den Schwerpunkten Daten- und Nachrichtentechnik. Thema seiner Diplomarbeit ist die vorgestellte Doppelprozessorkarte.
Hobbys: Motorrad-Oldtimer, Fotografie, Mikrocomputerei

Entwicklungsstand

Das beschriebene Multiprozessorsystem befindet sich zur Zeit am Institut für Regelungstechnik der TU Braunschweig in der Erprobung. Mit Hilfe der auf jeder Rechnerkarte vorhandenen Schnittstellen und der entwickelten Betriebssoftware (Disassembler, Single-Step-Programm für TMS 99000 und TMS 320) lassen sich effiziente Softwaretests durchführen [3, 4]. Die Installation der auf den einzelnen Systemeinheiten ablaufenden Regelprogramme ist hierdurch wesentlich erleichtert.

Literatur

[1] *Kratz, G.; Schumacher, W.:* Signalprozessor optimal an Mikrocomputer angekoppelt. ELEKTRONIK 1984, H. 6, S. 63...67.
[2] *Freund, E.; Hoyer, H.:* Das Prinzip nichtlinearer Systementkopplung mit der Anwendung auf Industrieroboter. Regelungstechnik 1980, H. 3 und 4.
[3] *Czarkowski, U.:* Monitorprogramme für TMS 99000. Entwurf am Institut für Regelungstechnik der TU Braunschweig, 1984.
[4] *Brendes, M.:* Debug-Programm für den Signalprozessor TMS 320. Entwurf am Institut für Regelungstechnik der TU Braunschweig, 1983.
[5] *Wetzel, W.:* Doppelprozessorkarte mit TMS 99000 und TMS 320. Diplomarbeit am Institut für Regelungstechnik der TU Braunschweig, 1984.
[6] TMS 99105/99110 16-Bit Microprocessor Data-Manual. Texas Instruments.
[7] TMS 320 Data-Manual. Texas Instruments.
[8] Multiprozessor-Systeme. ELEKTRONIK-Sonderheft Nr. 54.
[9] *Schumacher, Letas, Leonhard:* Können hochdynamische Drehstrom-Stellantriebe in Zukunft Gleichstrom-Antriebe ersetzen? KEM 1984, H. 4, S. 37...39.

Arbeitshilfe

Die ELEKTRONIK ist für Sie eine echte Arbeitshilfe, wenn Sie sich mit der Entwicklung und industriellen Anwendung elektronischer Schaltungen und Baugruppen von Geräten und Systemen befassen.



Die ELEKTRONIK liefert Ihnen Informationen zur Konzeption, Projektion, Entwicklung und über die Einführung elektronischer Systeme in die Fertigung.

Die ELEKTRONIK informiert Sie über Bauelemente:
● Speicher aller Art ● Mikroprozessoren ● Logik-Bausteine ● Interface-Bausteine ● Signalprozessoren ● Operationsverstärker ● A/D- und D/A-Umsetzer ● Leistungshalbleiter ● Passive Bauelemente ● Elektromechanische Bauelemente ● Optoelektronische Bauelemente ● Sensoren und Aktuatoren.

Die ELEKTRONIK informiert Sie über Meß- und Prüftechnik:
Die ELEKTRONIK beobachtet die Marktentwicklung, beschreibt technische Grundlagen, gibt Spezialinformationen und macht so den Markt für den Interessenten überschaubar.

Die ELEKTRONIK informiert Sie über Automatisierung:
Besonders intensiv behandelt wird das Gebiet der Klein- und Mittelserienfertigung. Vielfältige Beispiele zeigen, was Industrieroboter in der Automatisierung zu leisten vermögen.

Die ELEKTRONIK informiert Sie über Kommunikationstechnik:
Einige Stichworte:
● Technologien der elektronischen Kommunikation ● Bauelemente für die Telekommunikation ● Übertragungsverfahren ● Meßtechnik ● Normung ● Internationale Zusammenarbeit ● Neue Medien ● Pläne der Fernmeldebehörden.

Die ELEKTRONIK informiert Sie über Schaltungspraxis:
Hier werden erprobte und aktuelle Schaltungen, Entwurfs- und Berechnungshilfen, Meßideen und Anwendungsbeispiele aus dem professionellen und dem industriellen Bereich präsentiert. Beispiele:
● Allgemeine Digitaltechnik ● Interface-Schaltungen ● Signalerzeugung ● Meßtechnik ● Meßwerterfassung und Verarbeitung ● Stromversorgung ● Spezialschaltungen ● µC-Praxis ● Applikationen.

Die ELEKTRONIK informiert Sie über Mikrocomputer:
● Tischcomputer ● Mikroprozessor-CPUs (Bauelemente und Platinen) ● Einchipcomputer ● Mikroprozessor-Peripherie (Bauelemente und Geräte) ● Speicher (Systeme und Bauelemente) ● Software (Systemebene, Anwenderebene, Sprachen) ● Entwicklungssysteme ● Anwendungen, z. B. in der Steuerungstechnik, Medizinelektronik, Konsumelektronik, Meßtechnik ● Praxisnahe Hinweise für Entwickler (µC-Praxis) ● Meß- und Prüftechnik für Mikroprozessorsysteme.

Die ELEKTRONIK informiert Sie über das Produktangebot auf dem internationalen Markt:
● Bauelemente ● Meßgeräte
● Mikrocomputer ● Datentechnik
● Fertigungsmittel ● Software
● Steuer- und Regeltechnik
● Prüftechnik

Die ELEKTRONIK informiert Sie über Aktuelles in der Branche:
Kurzmeldungen aus aller Welt zum Thema ● Technologien ● Verfahren
● Neuheiten ● Unternehmen
● Termine ● Personen

Die ELEKTRONIK informiert Sie mit Sonderpublikationen als Heft im Heft über ● **CAD/CAM** Rechnerunterstützte Entwicklung und Fertigung ● **COM & PRO** Computer und Programme für Anwender von OEM-Produkten ● **TELECOM** Bausteine und Verfahren der Telekommunikation
● **ROBOTER** Flexible Automatisierung in der Industrie

Bitte verwenden Sie zum Kennenlernen unserer ELEKTRONIK nebenstehende Karte.



Fachzeitschrift für Entwickler und industrielle Anwender